CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduada en Matemáticas e Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Interconexión de centros de datos mediante técnicas SDN

Autor:
María Victoria Álvarez Sierra

Directores:
Gregorio Hernández Peñálver (UPM)
Luis M. Contreras Murillo (Telefónica I+D)

JUNIO, 2015

"No temas a las dificultades.

Lo mejor surge de ellas."

- Dra. Rita Levi-Montalcini.

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis padres por el esfuerzo que han puesto en darme la mejor educación y apoyarme siempre en todo lo que he hecho. Gracias por darme la posibilidad de estudiar la carrera que quise, aquella carrera que no era posible hacerla en Extremadura. Gracias por ayudarme en todo momento y por todo el cariño que me habéis demostrado siempre.

En segundo lugar, agradecer a mi pareja Javier por apoyarme y ayudarme siempre. Gracias por tener tanta paciencia conmigo en mis momentos de estrés, por aguantarme y por estar siempre a mi lado. Gracias a mi compañera Lidia por el apoyo que me ha demostrado siempre y por hacer mi vida universitaria más entretenida. Han sido 4 años conviviendo día a día que recordaré siempre.

También, quiero agradecer a todos mis profesores de la universidad por haberme enseñado tanto y por demostrarme lo apasionante que puede llegar a ser la informática. En especial a mi tutor Gregorio Hernández, por ser un gran tutor y profesor en la carrera, y por no perder nunca esa sonrisa al enseñar un mundo tan bonito como son las matemáticas.

Finalmente, quería agradecer también a mis tutores, Luis M. Contreras, Víctor López y Óscar González, por darme la posibilidad de realizar este proyecto y enseñarme tanto en tan poco tiempo. Ha sido un placer poder trabajar con todos vosotros.

A todos, gracias.

RESUMEN

En este trabajo final de grado se ha contribuido a la interconexión de centros de datos distribuidos geográficamente, integrando para ello nuevas funcionalidades en la arquitectura Application-Based Network Operations (ABNO) y configurando los componentes software necesarios. ABNO engloba distintas tecnologías que recogen la información sobre los recursos disponibles en la red con el objetivo de proporcionar rutas específicas para el tráfico.

La solución que se presenta en este trabajo se basa en las redes definidas por software (Software-Defined Networking, SDN), como solución innovadora para mejorar la gestión y el control de las infraestructuras que pertenecen a múltiples dominios administrativos, pero trabajan en colaboración en una federación común mejorando la calidad del servicio ofrecido. La conectividad entre los diferentes dominios es posible gracias a los Túneles GRE.

Cada centro de datos supone un dominio administrativo diferenciado, disponiendo cada uno de ellos del software de gestión en la nube OpenStack para la creación de las máquinas virtuales (VM) que posteriormente serán interconectadas. Además, cada centro de datos también contará con el controlador Ryu SDN que se encargará del control de la conectividad, siendo también independiente para cada uno de estos dominios.

Con el objetivo de mantener una visión integral de todos los recursos de la red disponibles, y de proporcionar una conectividad extremo a extremo (E2E) requerida por los centros de datos, la arquitectura ABNO ha tenido que ser modificada para soportar estas nuevas funcionalidades, así como validada en un escenario con infraestructuras multidominio.

PALABRAS CLAVE

ABNO, centro de datos, conectividad, dominios administrativos diferenciados, E2E, OpenStack, Ryu, SDN, Túneles GRE, VM.

ABSTRACT

This work contributes to the interconnection of geographically distributed data centers (DC), being necessary to integrate new features with the Application-Based Network Operations (ABNO) architecture and to configure the software components required. ABNO gathers different technologies together to store the topology network information and the available resources to provide paths to accommodate the traffic in the network.

The solution proposed in this document is based on Software-Defined Networking (SDN), as an innovative solution to improve management and control of multiple administrative domains, which can work together in a common federation in order to improve the quality of service. Connectivity between different domains is possible through GRE Tunnels.

Each DC represents a distinct administrative domain, and through the management doud software (OpenStack), it can create virtual machines (VM) that will be interconnected later. In addition, each data center will also incorporate the Ryu SDN controller, which is responsible for monitoring the connectivity. These Ryu controllers are also integrated in a separately way for each of these domains.

In order to keep an overview of the entire available network resources and to provide End-To-End (E2E) connectivity required by the data centers, ABNO architecture has been modified and tested in a multidomain infrastructure to support the new features implemented.

KEY WORDS

ABNO, connectivity, data center, distinct administrative domains, E2E, GRE Tunnels OpenStack, Ryu, SDN, VM.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS

ABNO	Application-Based Network Operations
API	Application Programming Interface
BoD	Bandwidth on Demand
DC	Data Center
DCN	Data Center Network
E2E	End-To-End
FI	Future Internet
FI-PPP	Future Internet Public-Private Partnership
GMPLS	Generalized Multiprotocol Label Switching
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
LSP	Label Switched Path
LSP-DB	Label Switched Path Database
MDVPN	Multi-Domain Virtual Private Network
NaaS	Network as a Service
NFV	Network Functions Virtualization
NREN	National Research and Education Network
ONF	Open Networking Foundation
OPEX	Operating EXpense
OVS	Open vSwitch
OVSDB	Open vSwitch DataBase
P2P	Peer-To-Per
PaaS	Platform as a Service
PCE	Path Computation Element
PCEP	Path Computation Element communication Protocol
QoE	Quality of Experience
QoS	Quality of Service
REST	REpresentational State Trasfer

SaaS	Software as a Service
SDN	Software-Defined Networking
SSH	Secure SHell
TED	Traffic Engineering Database
TIC	Tecnologías de la Información y la Comunicación
TID	Telefónica I+D
VLAN	Virtual Local Area Network
VXLAN	Virtual Extensible Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network

1. Introducción

1.1 Motivación

El imparable aumento del tráfico de datos y las nuevas necesidades de conectividad requieren tecnologías potentes que sean cada vez menos dependientes del hombre. Internet ha dado lugar a la creación de una sociedad digital, en la que casi todo está conectado y es accesible desde cualquier lugar. Sin embargo, a pesar de su uso generalizado, las redes IP tradicionales son complejas y difíciles de manejar [1], siendo bastante complicado configurar la red de acuerdo a las políticas predefinidas y reconfigurarla para que tenga la capacidad de responder a fallos y a cambios. Para hacer las cosas aún más complicadas, las redes actuales se encuentran integradas también verticalmente, juntándose el plano de control (software) y el plano de datos (hardware).

Software-Defined Networking (SDN) [2] es un paradigma emergente que promete cambiar este estado, rompiendo para ello la integración vertical, separando la lógica de control propia de la red de los routers y switches subyacentes, promoviendo la centralización lógica del control de la red e introduciendo la capacidad de programar la red, característica principal que diferencia a las redes SDN de las redes convencionales.

En este trabajo final de grado se contribuirá al proyecto europeo XIFI (eXperimental Infraestructures for the Future Internet), el cual representa a una federación de centros de datos (DC) distribuidos geográficamente y controlados por SDN. Con el objetivo de poder proporcionar servicio en la nube, se hace uso de un software de gestión especializado, OpenStack, para la creación de las máquinas virtuales (VM) en cada DC.

Una vez que las máquinas virtuales han sido creadas en cada uno de los centros de datos y se encuentran interconectadas entre sí, un servicio de conectividad de red tiene que ser invocado para la conexión de las redes superpuestas a través de la WAN (Wide Area Network).

Debido a que los DC mencionados representan dominios administrativos independientes, el despliegue de un solo controlador SDN en la federación para gestionar todas las conexiones necesarias no es suficiente. La existencia de diferentes dominios administrativos y un gran número de dispositivos complica la operación y compromete la capacidad de ampliación del controlador. Sin embargo, la arquitectura Application-Based Network Operations (ABNO) es capaz de mantener una visión integral de todos los recursos de red disponibles con el fin de orquestar todos ellos y proporcionar conectividad.

1.2 Objetivos

La finalidad de este trabajo fin de grado es el diseño, implementación y validación de nuevas funcionalidades en el controlador ABNO necesarias para establecer la interconexión de los centros de datos distribuidos geográficamente. Así como el despliegue y configuración de todo el software necesario para su funcionamiento.

Los objetivos principales son:

- Configuración del software necesario para proporcionar interconectividad entre los centros de datos.
    - **OpenStack**. Se usará para proporcionar servicio en la nube y la creación de las máquinas virtuales.
    - **Ryu**. Se define como un software definido en el marco de redes basado en componentes, facilitando las tareas de creación de nuevas aplicaciones de gestión y control de la red. Será imprescindible para proporcionar la conectividad extremo a extremo entre los nodos de la infraestructura.
    - **Túneles GRE**. Permitirán la conectividad entre los diferentes dominios creando una red privada punto a punto como si se tratara de una red VPN (Virtual Private Network)

- Diseño, implementación y validación de nuevas funcionalidades en ABNO. Se partirá del software ABNO implementado en Telefónica I+D (TID).
    - Integración de ABNO con OpenNaaS.
    - Integración de ABNO con OpenStack.
    - Integración de ABNO con Ryu.
    - Integración de ABNO con túneles GRE.

1.3 Estructura de esta memoria

La memoria se estructura como sigue, en la sección 2 se introduce el proceso de automatización de centros de datos, los problemas de interconexión y las posibles soluciones, entre ellas el caso XIFI. En la sección 3 se explica en detalle la solución de control XIFI y los componentes con los que interactúa: los controladores SDN locales en cada región, OpenNaaS, OpenStack y ABNO. El desarrollo e implementación de este último componente será expuesto con detalle en la sección 4, así como su integración y validación con OpenNaaS, RYU y OpenStack. Finalmente, se concluye la memoria y se muestra el posible trabajo futuro, sección 5.

2. Interconexión de Data Centers

2.1 Motivación

En los últimos años los centros de datos (DC) se han enfrentado a profundos avances: desde el almacenamiento y la computación orientada a sistemas, hasta poder alojar sitios web. Los DCs se han convertido en el componente básico de las arquitecturas de computación en la nube.

Actualmente, los servicios en la nube se consideran el punto central para la gestión de la enorme cantidad de datos de los usuarios e interconectar todas las máquinas que los producen. A su vez, todos estos servicios permiten extraer información importante que puede ser utilizada en muchos de los denominados escenarios "Smart-City". Por lo tanto, cada vez más, la computación en la nube es un componente indispensable para el denominado Futuro de Internet (FI).

El futuro de Internet (FI) es una infraestructura emergente de comunicaciones abiertas que facilita la oferta de una gama muy amplia de servicios y aplicaciones, estando de esta manera mejor acoplado Internet a los usuarios y al mundo físico.

Para dar forma a esta idea nace la iniciativa europea Future Internet Public-Private Partnership (FI-PPP) [3], un ambicioso programa por parte de la Comisión Europea que promueve la adopción de FI. Esta iniciativa tiene como objetivo explorar el potencial de una plataforma común para las futuras tecnologías de Internet y los ensayos a gran escala, con el fin de establecer nuevos ecosistemas empresariales y hacer frente a la oferta de servicios en toda Europa.

2.2 Problemas de interconexión de Data Centers

El enfoque convencional [4] para conectar recursos informáticos implica normalmente una reconfiguración manual cada vez que se implementa una nueva capacidad dentro de un dominio, creando ineficiencias y costes adicionales en la cadena.

Aunque la situación es accesible cuando se considera un dominio controlado por un único administrador de la infraestructura, la metodología resulta inviable en el caso de la configuración y el funcionamiento de un entorno distribuido, donde varias entidades administrativas se encuentran provisionando servicios entre dominios en una federación común.

Las infraestructuras basadas en la nube permiten a los usuarios finales implementar aplicaciones y servicios de una manera distribuida, haciendo para ello uso de los recursos informáticos asignados en los centros de datos distribuidos [5].

Estas comunicaciones entre los centros de datos se realizan mediante circuitos que ofrecidos por Operadores y Proveedores de Servicios (SP), los cuales suelen presentar tecnologías y topologías heterogéneas que comprenden diferentes dominios administrativos. Adicionalmente, los recursos de red pueden requerir ser reasignados dinámicamente de manera interna o entre diferentes centros de datos, cambiando sobre la marcha la superposición de la topología del servicio y la carga del tráfico correspondiente.

Por un lado, la prestación de servicios distribuidos en la nube implica la coordinación inter-dominio en términos de reenvío, encaminamiento, protocolos, niveles de calidad del servicio (QoS) y seguridad. Pero por otro lado, también existen aspectos intra-dominio importantes a considerar, tales como la recuperación de fallos o el control de la congestión.

Los principales requisitos de los servicios ofrecidos en los DCs, como son la escalabilidad, las arquitecturas multi-tenant (múltiples usuarios), la migración de las máquinas virtuales (VM) y la facilidad de configuración, han tenido que enfrentarse a un elevado número de problemas para el uso de las redes virtuales.

El enfoque tradicional para la conexión de los recursos informáticos a la red es utilizar la segmentación de red basada en VLAN (Virtual Local Area Networks) [6] con el objetivo de implementar el aislamiento entre los usuarios, pero este método presenta a su vez problemas. Si se escoge la segmentación de usuarios siguiendo VLAN, se produce una limitación de espacio de direccionamiento a 4096 (12-bit VLAN), de modo que la segmentación de la red proporciona un número limitado de redes configurables por cada DC, se delimita la segmentación de usuarios a 4096, no pudiendo añadir más.

Además, VLAN está demasiado ligada a la infraestructura física y su configuración es difícil de manejar, por lo que desde el punto de vista operativo es una solución muy poco automática, de hecho se requiere realizar un inventario donde quede almacenada todas las configuraciones de las VLAN correspondientes.

Por otra parte, a pesar de que las redes IP tradicionales son complejas y difíciles de manejar [1], los protocolos distribuidos de control y de red de transporte que se ejecutan dentro de los routers y switches son de uso generalizado, ya que son las tecnologías clave que permiten a la información, en forma de paquetes digitales, viajar por el mundo.

Para expresar las políticas de red de alto nivel deseadas, los operadores de red necesitan configurar cada dispositivo individual de red por separado, utilizando con frecuencia comandos de bajo nivel específicos del proveedor. Además de la complejidad de la configuración, los entornos de red tienen que soportar la dinámica de fallos y adaptarse a los cambios de carga. Los mecanismos automáticos de reconfiguración y respuesta son prácticamente inexistentes en las redes IP actuales. Por lo tanto, la aplicación de las políticas necesarias en un entorno tan dinámico es muy difícil.

Para hacerlo aún más complicado, las redes actuales también están integradas verticalmente. El plano de control (que decide cómo manejar el tráfico de red) y el plano de datos (que reenvía el tráfico de acuerdo con las decisiones tomadas por el plano de control)

se integran dentro de los dispositivos de red, lo que reduce la flexibilidad y obstaculiza la innovación y la evolución de la infraestructura de red.

Como idea de la dificultad de evolución de estos equipos integrados, cabe destacar el ejemplo de la transición de IPv4 a IPv6 que se inició hace más de una década y que todavía en gran parte está incompleta, cuando de hecho IPv6 representa simplemente una actualización de protocolo. Debido a la inercia de las redes IP actuales, un nuevo protocolo de enrutamiento puede tomar de 5 a 10 años para ser completamente diseñado, evaluado y desplegado [7], [8].

2.3 Posibles soluciones

Para resolver los problemas sobre la segmentación basada en VLAN se han propuesto varias soluciones, todas ellas con un enfoque similar. Algunas de estas ideas se centran en la concatenación de etiquetas IEEE 802.1ad Provider Bridges (Q-in-Q) [9], las cuales amplían la cabecera permitiendo múltiples etiquetas VLAN en una trama Ethernet (por ejemplo, dientes VLAN dentro de SP VLAN), mientras que otras ideas apuntan a que sea la solución IEEE 802.1ah Provider Backbone Bridges (MAC-in-MAC) [10] con encapsulación de la trama en otra cabecera Ethernet, permitiendo la posibilidad de interconectar múltiples redes de proveedores de switches.

A pesar de que estos criterios son la solución a algunos problemas, no representan la solución adecuada para las redes de centros de datos, ya que estas opciones sólo son válidas desde el punto de vista del plano de datos, manteniéndose la complejidad de configuración, la cual sigue siendo manual. Con el objetivo de resolver los problemas relacionados con la prestación de servicios a múltiples dientes en grandes DCNs, a continuación se presentan dos soluciones elementales: SDN y OpenFlow.

2.3.1 SDN

Las redes definidas por software [11], [12] permiten la gestión de un entorno complejo y heterogéneo de una manera simple, escalable y flexible. Las arquitecturas basadas en SDN se caracterizan por desacoplar la inteligencia de la red del plano de reenvío (forwarding), por lo que el control y el estado de la red pueden ser lógicamente externalizados por medio de un controlador SDN. De este modo, la infraestructura de la red posee dinamismo por parte de los servicios doud computing y define una interfaz capaz de interactuar con las funciones del plano de datos de los nodos de la red.

La separación de las preocupaciones introducidas entre la definición de políticas de red, su implementación en hardware de conmutación, y el reenvío de tráfico, es clave para la flexibilidad deseada: rompiendo el problema de control de la red en trozos manejables, SDN hace que sea más fácil crear e introducir nueva abstracciones en la creación de redes, lo que simplifica la gestión de redes y facilita la evolución de la red.

La separación del plano de control y el plano de datos se puede realizar por medio de una interfaz de programación bien definida entre los conmutadores y el controlador SDN [13].

El controlador ejerce un control directo sobre el estado de los elementos del plano de datos a través de esta interfaz de programación de aplicaciones bien definida (API), como se muestra en la Figura 1.

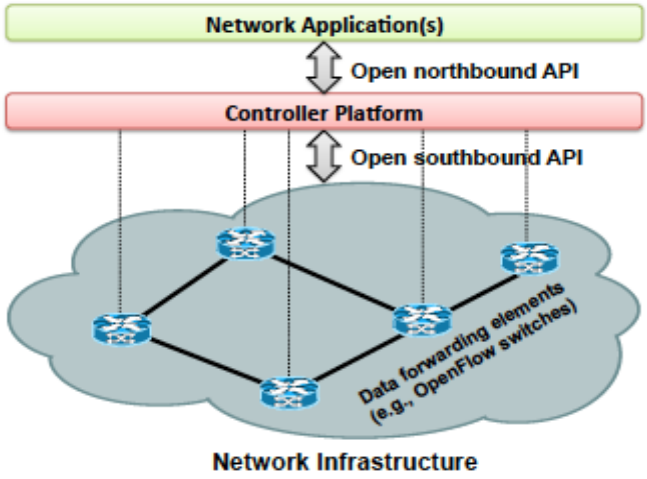

Fig. 1 Vista simplificada de una arquitectura SDN.

El controlador SDN es una entidad lógica centralizada y responsable de un conjunto de tareas, incluyendo la extracción y el mantenimiento de una visión global de la topología de red y de su estado, así como de la creación de instancias lógicas de reenvío adecuadas para un determinado escenario.

En la práctica, este controlador gestiona las conexiones a todos los elementos de la red, de manera que se encarga de instalar, modificar y eliminar las entradas de reenvío de las tablas de los switches conectados, mediante el uso de mensajes de control específicos del protocolo.

2.3.2 OpenFlow

La comunicación descrita anteriormente entre los controladores y los elementos de red en la arquitectura SDN se basa normalmente en el protocolo OpenFlow [11], el cual se originó en la universidad de Stanford y actualmente es mantenido por la Fundación Open Networking (ONF). Hasta la fecha, OpenFlow es el estándar de conducción dominante para las redes SDN.

Haciendo uso de OpenFlow [14], el controlador puede dictar normas específicas a los switches habilitados para SDN (Figura 2). Estas reglas definen si los flujos que se ajustan a las características específicas deberán ser remitidos, desviados, modificados o caídos en forma de QoS.

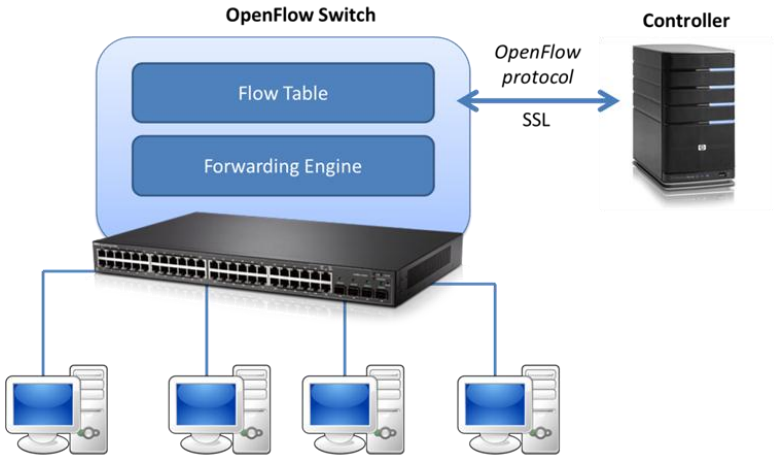

Fig. 2 Interacción entre SDN y Openflow.

Un switch OpenFlow [15], [16] consiste en una o más tablas de reglas de manejo de paquetes (tabla de flujos), de manera que cada regla coincide con un subconjunto del tráfico y realiza ciertas acciones (reenvío, modificación, etc.) sobre el tráfico. Dependiendo de las reglas instaladas por el controlador de la aplicación, un switch OpenFlow puede, instruido por el controlador, comportarse como un router, switch, firewall, o llevar a cabo otras funciones como balanceador de carga.

A través de OpenFlow, el controlador SDN puede definir reglas para los flujos de paquetes individuales de acuerdo a los campos específicos de la cabecera del paquete en cualquiera de las capas 2 o 3 [4]. Estas reglas se muestran en forma de entradas en la tabla de flujo que resulta del reenvío de la configuración de los nodos. Esta sofisticada manipulación del flujo se puede lograr mediante una conexión en cascada de un cierto número de tablas de flujo, las cuales actúan sobre el tráfico antes de que las reglas se entreguen por el puerto de salida.

Esta granularidad en el control proporciona una alta flexibilidad. OpenFlow implementa una interfaz abstracta independiente del proveedor, que ayuda a integrar las redes heterogéneas de múltiples fabricantes. Al mismo tiempo que facilita la orquestación de recursos de la red, ya que la representación del comportamiento nodo se convierte en genérico e independiente de las implementaciones particulares del hardware y de la sintaxis del comando.

En numerosos entornos distribuidos con un gran número de dominios administrativos y recursos, no es suficiente la implementación de un único controlador SDN para gestionar todas las conexiones necesarias. Una posible solución podría consistir en la federación de dominios basados en SDN no superpuestos. Sin embargo, la existencia de múltiples dominios incrementa el problema de la coordinación del controlador para la prestación de un servicio de conectividad de extremo a extremo.

La interconexión de controladores SDN se ha propuesto como una manera [17] de intercambiar información entre ellos. Sin embargo, en un entorno tan federado se requiere una visión lógica centralizada y un aprovisionamiento orquestado de todos los recursos de la red para establecer la conectividad extremo a extremo (E2E) entre los servicios de los centros de datos.

2.4 XIFI

Dentro de este marco, el proyecto europeo XIFI [18] tiene como objetivo [5] ser la comunidad nube para futuros desarrolladores de servicios de Internet, proporcionando un mercado común de componentes para las aplicaciones a través de la creación de una red europea de infraestructuras federadas. Al estar fuertemente centrado en los servicios basados en doud computing y arquitecturas en la nube, XIFI aprovecha un conjunto de instalaciones de centros de datos distribuidos geográficamente por toda Europa, e interconectados por medio de los servicios de red proporcionados por Redes Nacionales de Investigación y Educación (National Research and Education Network, NREN), los cuales están interconectados gracias a la red paneuropea de investigación y educación GÉANT.

El desarrollo de este proyecto XIFI viene motivado por el crecimiento de las aplicaciones y servicios de internet que necesitan conectividad de red y reserva de recursos. Esto se logra gracias a la tecnología FI-WARE [19], la cual persigue desarrollar una plataforma de middleware abierta sobre la que poder desarrollar las nuevas aplicaciones y los servicios de Internet del Futuro.

FI-WARE proporciona tecnologías que permiten la explotación de Datos Abiertos (Open Data), de modo que pone a disposición de los desarrolladores de aplicaciones las tecnologías que permitan explotar esos datos. Otra de las ventajas que aporta esta tecnología es la rápida adopción por partes de la ciudades, de modo que uno de los

dominios más relevantes para su aplicación es el asociado a las Ciudades Inteligentes (Smart Cities).

La federación XIFI se define [4] como un compendio de infraestructuras de pruebas, también conocidas como nodos o regiones, pertenecientes a organizaciones administrativas independientes, que de una manera colaborativa proporcionan servicios multi-dominio de computación en la nube.

La comunidad inicial de infraestructuras crea una federación totalmente funcional para que otros nodos potenciales pueden unirse y formar parte de ella. Luego es necesario una definición de los requisitos operativos y técnicos que deben cumplir los nodos que deseen unirse, así como los procedimientos de apoyo para ayudar a estos nodos en la integración y despliegue dentro de la federación existente. De esta forma, se logra que la federación sea un entorno flexible para la unión de posibles nodos que hayan alcanzado el nivel mínimo de cumplimiento y que están listos para un cierto compromiso con los principios y objetivos de la federación.

Desde una perspectiva técnica, se consideran tres tipos de capacidades de infraestructura:

- **Capacidad de computación:** las infraestructuras que proporcionan el hospedaje y la provisión de los recursos software (por ejemplo, los centros de datos).
- **Capacidad de datos:** las infraestructuras que proporcionan las fuentes de datos que se pueden conectar a las aplicaciones (por ejemplo, ciudades inteligentes o redes de sensores).
- **Capacidad de transporte:** las infraestructuras que proporcionan conectividad para apoyar la provisión de servicios y el acceso a los datos y usuarios (por ejemplo, a través de NREN).

Hasta el momento, la infraestructura XIFI comprende 19 nodos [3], [14], [18], repartidos por un total de 12 países, para hacer frente a pruebas y servir como escenario para diversas necesidades de conjunto de usuarios FI y experimentadores.

En la siguiente figura se pueden ver la distribución geográfica de los nodos actuales, así como el dominio español situado en Málaga y que se conecta a la federación a través de RedIRIS.

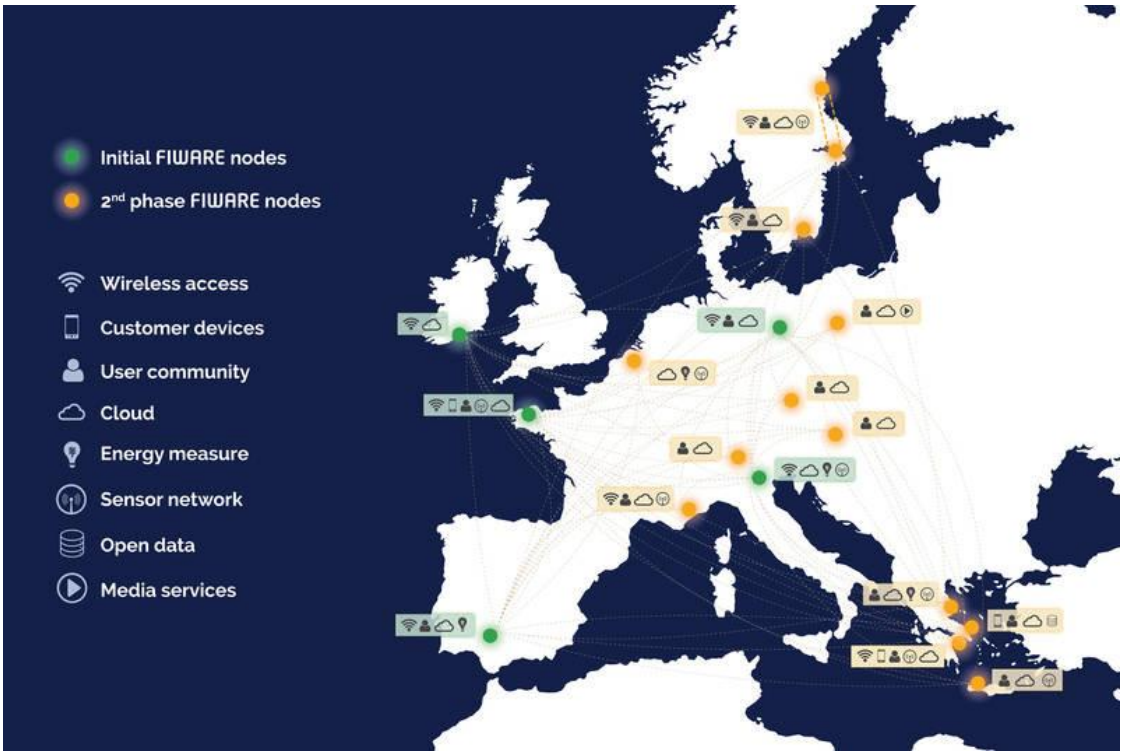

Fig. 3 Infraestructura federada XIFI.

En esta infraestructura, los recursos informáticos se conectan a conmutadores habilitados con OpenFlow, constituyendo un punto de demarcación para el controlador de red XIFI. Estos switches OpenFlow deberán ser instruidos por el controlador de red XIFI para la implementación del servicio de extremo a extremo (E2E). Desde el punto de demarcación a la WAN, una serie de servicios de transporte son pre-aprovisionados con el fin de ser seleccionada la WAN dinámicamente por los algoritmos controladores de red XIFI, de acuerdo a las peticiones de los usuarios finales.

En particular, la solución de gestión seleccionada por las capacidades de computación de cada DC se basa en OpenStack [14]. Actualmente, los sistemas de gestión en la nube como OpenStack permiten crear redes superpuestas en una sola infraestructura con una variedad de tecnologías (túneles GRE, VXLAN y VLAN). Desgraciadamente, estas capacidades de red se limitan a los dispositivos gestionados directamente por una sola instancia del sistema, que normalmente se encuentra una región única. Sin embargo, en el escenario de la federación, los recursos de información se distribuyen en ubicaciones geográficamente dispersas que son administradas por separado, formando diferentes dominios de la infraestructura.

SDN se propone como mecanismo de control capaz de coordinar de manera autónoma los recursos presentes en cada nodo de la federación, permitiendo la realización de un servicio unificado de operación, control y gestión. Sin embargo, el despliegue de la federación propuesta con un solo controlador SDN responsable de gestionar todas las conexiones necesarias es un enfoque que presenta limitaciones.

La existencia de diferentes dominios administrativos y un gran número de dispositivos, requiere de una entidad lógicamente centralizada, capaz de mantener una visión integral de todos los recursos de red disponibles. Por lo tanto, con el fin de permitir la composición de servicios extremo a extremo, se define un controlador de red XIFI, el cual orquestará los correspondientes servicios de conectividad en la federación mediante la interacción con los controladores locales SDN.

3. Solución de control

3.1 Arquitectura de alto nivel

La arquitectura XIFI se compone [5] de una federación de centros de datos distribuidos, los cuales ofrecen una infraestructura compartida de servicios basada en la nube, convirtiéndose en factor esencial la existencia de un cierto nivel de abstracción con el objetivo de proporcionar una visión común a los usuarios finales. Esto se consigue [21] con el modelo tradicional de plataformas en la nube: SaaS (Software as a Service), PaaS (Platform as a Service) e IaaS (Infraestructura como Servicio).

El concepto de SaaS se define como un modelo de distribución de software donde el soporte lógico y los datos manejados se alojan en servidores de una compañía de tecnologías de la información y la comunicación (TIC), a los que el consumidor podrá acceder gracias a un diente. Por otro lado, PaaS se refiere a la provisión de una plataforma informática y al despliegue del conjunto asociado de aplicaciones software. Por último, IaaS [21] se caracteriza por ser un conjunto de recursos puestos a disposición de los consumidores en la nube a través de un conjunto de interfaces. El consumidor estará provisto de una interfaz para gestionar la solicitud y la liberación de los recursos, mientras que no tendrá acceso físico de bajo nivel a ellos. Además, el consumidor podrá utilizar estos recursos como desee sin tener que preocuparse por el mantenimiento físico.

Este enfoque permitirá a la infraestructura de red acompasar el dinamismo de los servicios de computación en la nube, de manera que XIFI puede ser visto como una nube comunitaria [21], donde los usuarios finales no tienen que asumir ninguna negociación específica con XIFI para la provisión de conectividad de red, implementándose las aplicaciones y los servicios a través de los centros de datos distribuidos. Luego se requiere

de un sistema de aprovisionamiento integrado para automatizar la configuración de conectividad localmente en los distintos centros de datos. Para lograr este objetivo, se ha diseñado un controlador de red encargado de orquestar el proceso completo de aprovisionamiento de servicios.

Un controlador SDN, por sí solo, no es capaz de implementar las capacidades [5] de gestión de red necesarias para apoyar la conectividad distribuida del marco federado XIFI, así como hacer frente a la integración de los entornos de red heterogéneos previstos.

Es necesario un controlador de red XIFI que interactúe de manera inteligente con los controladores SDN desplegados a nivel local en cada centro de datos, como parte de la federación de dominios basados en SDN. Además, el controlador de red XIFI complementará los controladores SDN locales proporcionando la coordinación necesaria entre los múltiples sistemas de nubes independientes que residen en los diferentes dominios y proporcionando conectividad a través de las diferentes redes.

Por lo tanto, el controlador de red XIFI desencadena el aprovisionamiento de la prestación de servicios E2E de acuerdo a las necesidades y requerimientos del usuario final, mediante la configuración de una serie de nodos de red basados en SDN.

Una automatización completa es la clave para que sean rápidos el despliegue, el consumo y la gestión de servicios en la nube. Por otra parte, en escenarios basados en SDN la inteligencia de la red y el estado son centralizados de manera lógica, y la infraestructura de red subyacente se abstrae de las aplicaciones. Como resultado, los usuarios finales son capaces de definir y de abstraer los requisitos de los servicios de forma independiente de las tecnologías de red subyacentes.

Recursos de conmutación específicos serán desplegados como parte de la red XIFI, actuando como puntos de demarcación de los servicios de conectividad, los cuales serán manejados totalmente por el controlador de red XIFI mediante OpenFlow. Cada centro de datos poseerá su propio controlador OpenFlow independiente. Sin embargo, tal controlador de OpenFlow sólo se utilizará para fines de aprovisionamiento (instruyendo las reglas en los conmutadores) y para alimentar la información topológica.

Estos switches XIFI basados en OpenFlow se conectarán a la infraestructura de las redes nacionales, permitiendo así prestar servicios de conectividad inter-DC E2E. Con el fin de separar y aislar los diferentes flujos de tráfico de los usuarios finales, el servicio de conectividad es ofrecido mediante redes superpuestas (overlay) sobre la infraestructura física entre los puntos de demarcación XIFI.

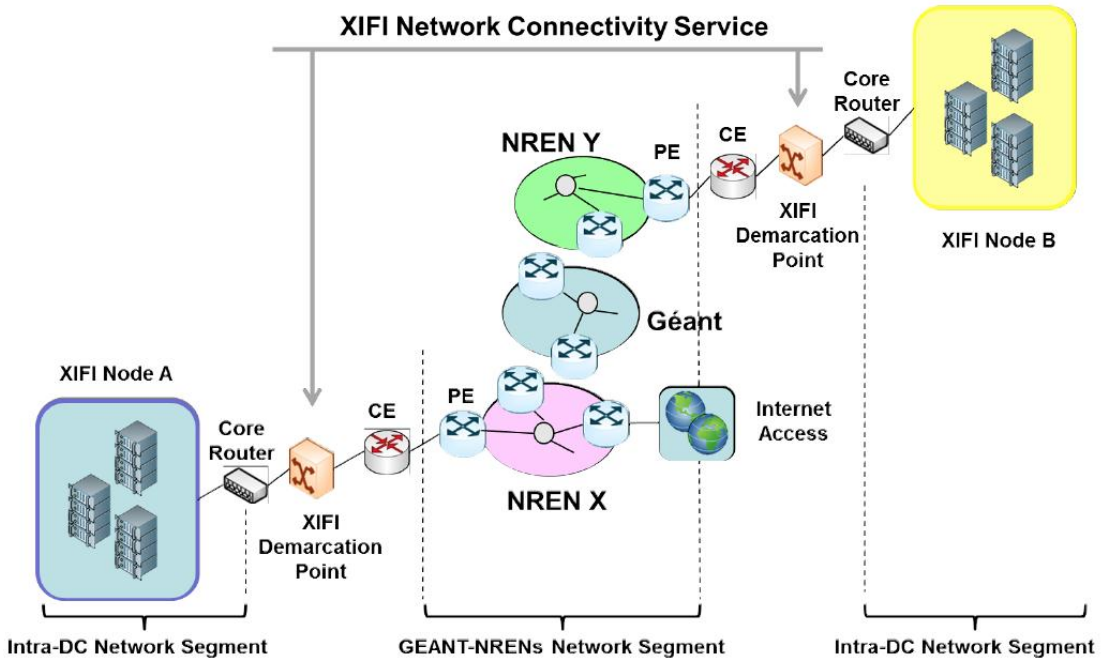

Fig. 4 Red Global XIFI.

La comunidad paneuropea de investigación y educación GÉANT y la comunidad NREN representan un excelente escenario para el despliegue de los servicios de conectividad XIFI, ya que ambas son proveedores de servicios de Internet especializados dedicados a apoyar las necesidades de las comunidades de investigación y educación. Por lo tanto, XIFI aprovecha este tipo de infraestructuras de red para llevar a cabo el servicio de conectividad de red E2E completo (Fig. 4).

Estos segmentos comprenden la conectividad intra-DC, la conectividad punto a punto (P2P) proporcionada por GÉANT y NREN y los denominados puntos de demarcación XIFI. Debido a las limitaciones tecnológicas actuales y la heterogeneidad de NREN, el servicio de conectividad de red XIFI específico es ofrecido entre los puntos de demarcación.

Los servicios E2E (End-To-End) de NREN implican un conjunto de particularidades específicas que hay que tener en cuenta, ya que influyen en general en el servicio de red XIFI:

- Los dominios de red NREN y GÉANT son capaces de proporcionar servicios de conectividad de capa 2 con multidominios E2E.
- La capacidad QoS E2E se basa en las capacidades de aprovisionamiento de NREN y GÉANT.

- La conexión a Internet debe realizarse en las instalaciones de NREN, ya que poseen rangos de IP pública.
- Para la conectividad NREN intra-dominio, siempre que sea posible, se utilizará mecanismos de aprovisionamiento automático como GÉANT BoD (Bandwidth on Demand); en caso contrario, la conectividad necesitará ser pre-definida y pre-aprovisionada de antemano.

La gestión de servicios de conectividad de red XIFI se lleva a cabo por medio de dos componentes clave de software. Por un lado, la gestión de servicios XIFI se realiza aplicando las extensiones necesarias para la plataforma OpenNaaS [22], [23]. Por otro lado, la gestión de instalaciones de la red se realiza a través del marco ABNO [20]. La coordinación entre estas dos entidades resulta crucial para una implementación adecuada del servicio de conectividad de red XIFI.

3.2 OpenNaaS

3.2.1 Introducción a NaaS

El servicio de conectividad XIFI [5] se enmarca dentro del concepto NaaS, ya que tiene como objetivo proporcionar un servicio de red E2E entre los nodos XIFI. Las capacidades de un modelo NaaS son interesantes con respecto al beneficio que pueden proporcionar a las comunidades GÉANT y NREN, permitiendo así un control total, la facilidad de la gestión y el funcionamiento de los recursos de red subyacentes, lo que conlleva un aumento de la flexibilidad y una reducción del OPEX (OPerating EXpense) en el funcionamiento del servicio de conectividad XIFI.

NaaS se define como un modelo de negocio relacionado con la red de servicio, cuyo objetivo es proporcionar servicios de red sobre infraestructuras proveedoras de servicios e Internet, con acceso directo, dinámico, escalable, seguro y aislando al inquilino de la red. El modelo NaaS abstrae la complejidad de la red subyacente y ofrece funciones y capacidades de la red como un servicio, permitiendo la posibilidad de desplegar y operar servicios de red avanzados fácilmente y ofrecer un rendimiento fiable y previsible de acuerdo a las necesidades de aplicación en la parte superior.

El concepto NaaS representa un interesante modelo de servicio en el que los clientes potenciales (experimentadores, instituciones de investigación) y los proveedores de servicios (por ejemplo NREN) se pueden beneficiar. Por otro parte, las organizaciones deben tener la capacidad de gestionar cualquier parte de la infraestructura y de los recursos

que reciben, así como la capacidad de desplegar sobre ellas cualquier tipo de aplicación o servicio.

Las capacidades de NaaS representan también un desafío para el desarrollo de servicios de virtualización de valor añadido, ya que pueden extender el modelo IaaS de centros de datos para el acceso y las redes de transporte. Además, NaaS puede permitir la evolución de la gestión de la red y de las operaciones, con el fin de aumentar la flexibilidad en las operaciones y reducir potencialmente los gastos operativos en un plazo corto/medio.

3.2.2 Módulo OpenNaaS

Una de las funcionalidades del marco NaaS consiste en abstraer la complejidad de la red subyacente, permitiendo así desplegar y operar fácilmente servicios de red avanzados de acuerdo a las peticiones de los usuarios finales. Estas capacidades permiten la extensión del modelo a partir de los centros de datos IaaS de acceso y las redes de transporte.

El modelo de NaaS se introduce en el proyecto XIFI con la plataforma OpenNaaS, con el objetivo de mejorar el servicio de conectividad de la red XIFI. OpenNaaS [23] es un framework de código abierto, desarrollado bajo el proyecto FP7 MANTYCHORE [24], que ofrece las herramientas necesarias para la gestión de los diferentes recursos presentes en cualquier infraestructura de red.

OpenNaaS se basa en un modelo operacional [5] abstracto y ligero, con una flexibilidad suficiente para adaptarse a diferentes módulos y proporcionar a los desarrolladores las herramientas necesarias para controlar la red XIFI detrás de los sistemas distribuidos. Además, el marco OpenNaaS permite la implementación de la lógica de un controlador SDN y de un plano de gestión en la parte superior del modelo abstracto, lo que se puede aplicar al control de los puntos de demarcación basados en OpenFlow en los nodos XIFI.

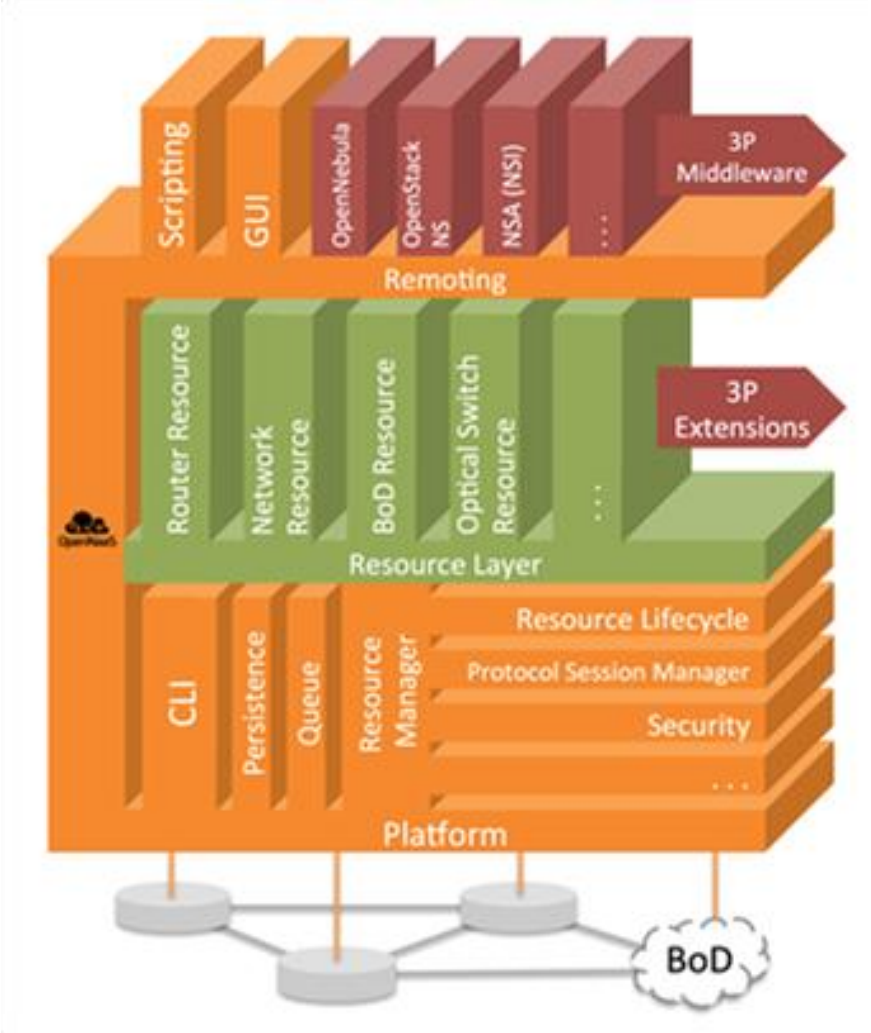

Fig. 5 Arquitectura OpenNaaS en capas.

La figura 5 muestra la arquitectura OpenNaaS, la cual se estructura en tres capas horizontales diferenciadas: la capa de plataforma (Platform) contiene los módulos reutilizables que forman el núcleo del software y controla el acceso a los diferentes recursos de la infraestructura; la capa de recursos (Resource Layer) contiene las diferentes abstracciones de recursos y define el conjunto de capacidades asignadas a las acciones que se pueden realizar sobre el recurso gestionado. Por último, la capa superior (Remoting), donde reside la inteligencia de la red, proporciona integración con aplicaciones externas e implementa las interfaces específicas hacia elementos externos. Por lo tanto, el marco OpenNaaS proporciona un marco adecuado para la gestión y operación de la parte superior de los servicios desarrollados, como el servicio de conectividad de red XIFI.

El módulo OpenNaaS es requerido [25] por el controlador de red XIFI para proporcionar conciencia de servicio de conectividad a través de la federación. Es el encargado de recoger y mantener la información sobre los recursos comprometidos para el usuario final desde el punto de vista de la conectividad, ya sea si los recursos son locales de un solo nodo XIFI o si se distribuyen en diferentes nodos de la federación. Por lo tanto, OpenNaaS se encargará de mantener la noción final de servicio de conectividad, asociando para ello las máquinas virtuales ubicadas en los DC separados y estableciendo la conexión de red necesaria entre ellos.

3.3 ABNO

3.3.1 Función del ABNO en el XIFI

Debido a que el proyecto XIFI es un representante de una federación controlada por SDN, la entrega de servicios distribuidos en la nube implica la configuración de varias capacidades a través de las redes involucradas en el servicio, tanto en los distintos centros de datos como en los dominios WAN conectados.

Dentro de cada centro de datos, la configuración del servicio en la nube para la creación de las máquinas virtuales se hace usando el software de gestión en la nube OpenStack. Una vez que las máquinas virtuales han sido creadas en cada DC y están internamente conectadas, un servicio de conectividad de red tiene que ser invocado para la conexión de las redes superpuestas a través de la WAN.

El enfoque de red [5] en XIFI basado en SDN sigue la arquitectura ABNO, propuesta en IETF [20] como un marco SDN basado en el estándar de bloques funcionales para hacer frente a la reserva específica de la aplicación de la conectividad de red, confiabilidad y recursos, interactuando a través de interfaces estándar. Los componentes del marco ABNO permiten una serie de funciones para actuar de una manera integrada, incluyendo el control de políticas, la información de la topología y la red aprovisionamiento de recursos.

El controlador ABNO es el responsable de orquestar las solicitudes de servicio de conectividad invocando a los componentes necesarios en el orden correcto de acuerdo a los flujos de trabajo específicos. Está conectado con el componente OpenNaaS, desde donde se progresa la solicitud de conectividad y posteriormente el módulo ABNO se encargará del enrutamiento y de la orquestación, instruyendo a los controladores SDN locales en cada uno de los nodos XIFI para poblar reglas de reenvío OpenFlow. De esta manera se consigue que los controladores locales actúen de una forma coordinada, estableciendo así el camino de conectividad y proporcionando una topología local en el interior de cada centro de datos.

Todo esto es posible debido a que el componente ABNO tiene una visión centralizada de la topología y de los recursos OpenFlow disponibles en cada uno de los nodos XIFI, como pueden ser las direcciones MAC o los puertos en switches físicos o virtuales entre otros.

A continuación se explicarán todos los componentes que han sido necesarios para poder utilizar ABNO en infraestructura XIFI.

3.3.2 Arquitectura ABNO

El Aplication-Based Network Operator (ABNO) se define como una arquitectura de red compuesta por varios módulos y tecnologías que obtienen información acerca de la topología y lo recursos disponibles en la red, actuando como un controlador/orquestador con el objetivo de proporcionar rutas para el tráfico y de reservar los recursos necesarios para ellas.

El marco ABNO se encuentra definido en la RFC 7491 [20] como una agrupación de componentes estándares y protocolos capaces de controlar y gestionar la red de forma coordinada, teniendo en cuenta las peticiones de aplicaciones externas. Los procedimientos y funcionalidades de esta potente arquitectura son los siguientes:

- Políticas de control de las entidades y aplicaciones para gestionar las peticiones que requieren el uso de recursos de red o de conexiones establecidas sobre ella.
- Reunir información sobre los recursos disponibles en la red.
- Reunir información sobre los recursos de red multicapa y cómo se distribuyen las topologías.
- Manejo de las peticiones y las respuestas de cálculos de caminos.
- Provisión y reserva de recursos de la red.
- Verificación de las conexiones y los recursos de instalación.

Como ya se ha mencionado, la arquitectura ABNO está compuesta por una serie de componentes, protocolos y procedimientos bien definidos, siendo capaz de combinarlos en una única arquitectura para dar servicio a los nuevos requisitos de demanda.

Es importante destacar, que no todos los módulos propuestos en marco IETF mencionado anteriormente se utilizan para el XIFI. La siguiente figura muestra una vista esquemática de los componentes de la arquitectura ABNO considerados para este proyecto.

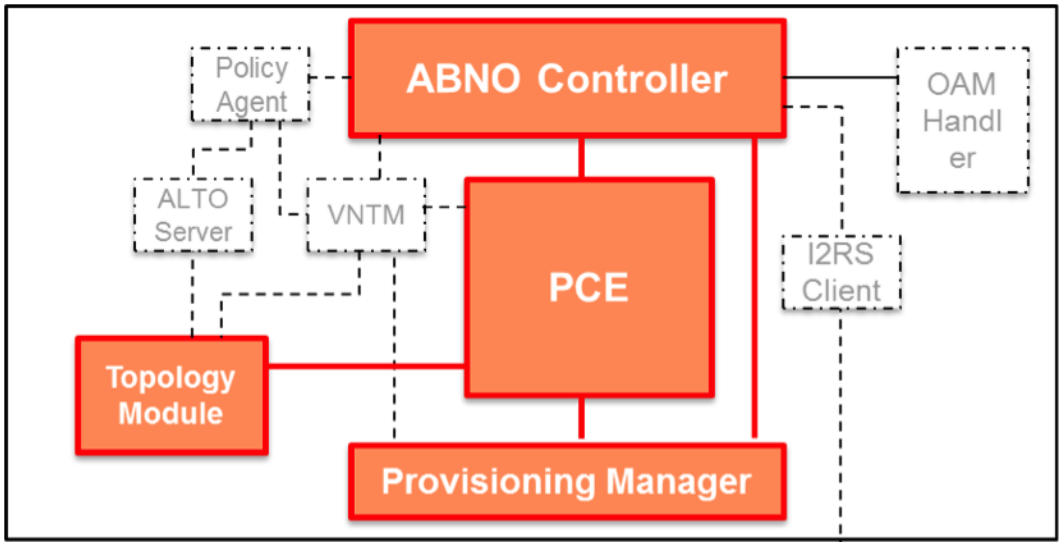

Fig. 6 Módulos del marco IETF ABNO considerados para el proyecto XIFI.

En las siguientes sub-secciones se describen los componentes funcionales que forman la arquitectura ABNO y se han utilizado para este proyecto, así como las interacciones entre ellos.

3.3.2.1 ABNO Controller

El controlador ABNO es el encargado de regular el comportamiento de la red en respuesta a posibles cambios en las condiciones de ésta, a nuevos requerimientos o a políticas de las aplicaciones de red. Es el punto de unión y de coordinación de las acciones individuales del resto de módulos de la arquitectura ABNO, invocando a los componentes implicados en el orden requerido.

3.3.2.2 Path Computation Element (PCE)

El Path Computation Element (PCE) se introduce en la RFC 4655 [26]. Es un componente funcional que computa rutas por la red en respuesta a peticiones, considerándose el componente principal de la arquitectura ABNO.

El PCE realiza los cálculos necesarios para encontrar la conexión entre los elementos de red en base a la topología de red que está almacenada en la Traffic Engineering Database (TED), la ingeniería de tráfico y el amplio conjunto de restricciones, incluyendo por

ejemplo el ancho de banda, la QoS y las exclusiones. Varios PCEs pueden colaborar para proporcionar caminos en un escenario de multidominio o en redes multicapa. Estos PCEs calcularían rutas basándose en TEDs locales.

3.3.2.3 Topology Module

El Topology Module mantiene una vista actualizada del gráfico de la topología, para ello incluye bases de datos que contienen información útil para el sistema. Las dos principales bases de datos son la Traffic Engineering Database (TED) y la Label Switched Path Database (LSP-DB). La TED almacena la información de los recursos de la red indicando cuáles están disponibles y cuáles no, pudiendo contener información sobre métricas o capacidad de ancho de banda de los enlaces entre otros. Por otra parte, la LSP-DB guarda información sobre los LSPs (Label Switched Path) que están activos en la red o que podrán establecerse.

3.3.2.4 Provisioning Manager

El Provisioning Manager es el responsable de dirigir las peticiones para el establecimiento de los LSPs. Este establecimiento lo puede hacer mandando instrucciones hacia el plano de control o programando directamente los dispositivos de la red. En este último caso, el Provisioning Manager que puede actuar como un OpenFlow Controller.

Estas operaciones pueden llevase a cabo en dos niveles:

- Peticiones para configurar recursos específicos en el plano de datos.
- Peticiones que desencadenan una serie de acciones que serán programadas con la asistencia del plano de control.

3.4 OpenStack

3.4.1 Introducción al Cloud Computing

El Cloud Computing [14] ofrece una gran flexibilidad para cambiar dinámicamente la topología del servicio y la demanda de tráfico correspondiente, impactando en las normas de planificación y en las reglas de dimensionamiento convencionales de los operadores de red. La integración eficiente de los servicios basados en la nube entre los DCs distribuidos, incluyendo la red de interconexión, se convierte en un reto con el fin de ofrecer garantías de rendimiento, localización y propiedades de alta disponibilidad.

El objetivo principal de la virtualización de red es permitir a múltiples dientes que la privacidad, el aislamiento y la fiabilidad están asegurados con el mismo grado que lo estarían en una infraestructura física dedicada. Con la separación lógica de los servicios se logra su protección con respecto a los de otros usuarios, en el sentido de no interferir con los procedimientos operacionales y el tráfico transportado.

Las actuales tecnologías de Cloud Computing permiten que los recursos de las VMs sean compartidos dinámicamente. Al encapsular el tráfico de las máquinas virtuales en un administrador de recursos en la nube, se pueden migrar cargas de trabajo de un DC a otro, con la correspondiente mejora de la calidad de la experiencia (QoE) percibida y la reducción del consumo de energía, o incluso mejora en la respuesta ante situaciones de fallos en la red o eventos de alta demanda. Además, al colocar las VMs más cerca de los usuarios finales, se permite el desarrollo de servicios y aplicaciones a muy baja latencia.

Para poder ofrecer servicios E2E a través de dominios heterogéneos, los centros de datos en la nube tienen que aprovechar los mecanismos de orquestación que permite la gestión conjunta de los DCs (almacenamiento y procesamiento) y la red, tanto en infraestructura como a nivel de servicio.

En este contexto, el Cloud Computing no debe considerarse como una sola tecnología. En su lugar, este concepto se basa en la aplicación sinérgica de tecnologías específicas que permiten:

- Virtualización de la infraestructura (DC y Red).
- Establecimiento de una red programable (SDN).
- Virtualización de las funciones de red (NFV).
- Gestión de los recursos federados y orquestación.

Por lo tanto, las plataformas de gestión en la nube son herramientas que proporcionan la gestión de entornos en la nube integrados e incorporan interfaces de servicio, a través de las cuales se puede solicitar una infraestructura virtual. Esta solicitud se emite a un controlador de la nube, el cual provisiona la correspondiente infraestructura virtual con los recursos disponibles dentro cada DC.

Una de las plataformas más populares de gestión de nube es OpenStack, la cual se detallará a continuación y será utilizada para este proyecto.

3.4.2 Arquitectura OpenStack

OpenStack [27] (Open Source Computing Cloud Computing Software) se define como un proyecto de computación en la nube que proporciona una infraestructura como servicio (IaaS). Se trata de una "sistema operativo nube" que controla grandes volúmenes de

almacenamiento, computación, redes y recursos, todo ello gestionado a través de un panel de control, ofreciendo a los administradores la posibilidad de controlar y gestionar la provisión de recursos a través de una interfaz web.

OpenStack es una plataforma de gestión en la nube de código abierto, desarrollada y apoyada por una colaboración mundial de desarrolladores y tecnólogos del Cloud Computing. Este software busca ofrecer soluciones para todo tipo de nubes por ser fácil de implementar y escalable. La tecnología consiste en una serie de proyectos interrelacionados que entregan diversos componentes para una solución de infraestructura de nube. Todo el código fuente de OpenStack está disponible [27] bajo una licencia Apache 2.0.

OpenStack está constituido por un diseño modular que permite la integración con las tecnologías existentes y de terceros. Está construido sobre una arquitectura basada en mensajería distribuida con componentes modulares, cada una de las cuales gestiona un servicio diferente; estos servicios, en conjunto, constituyen una nube IaaS.

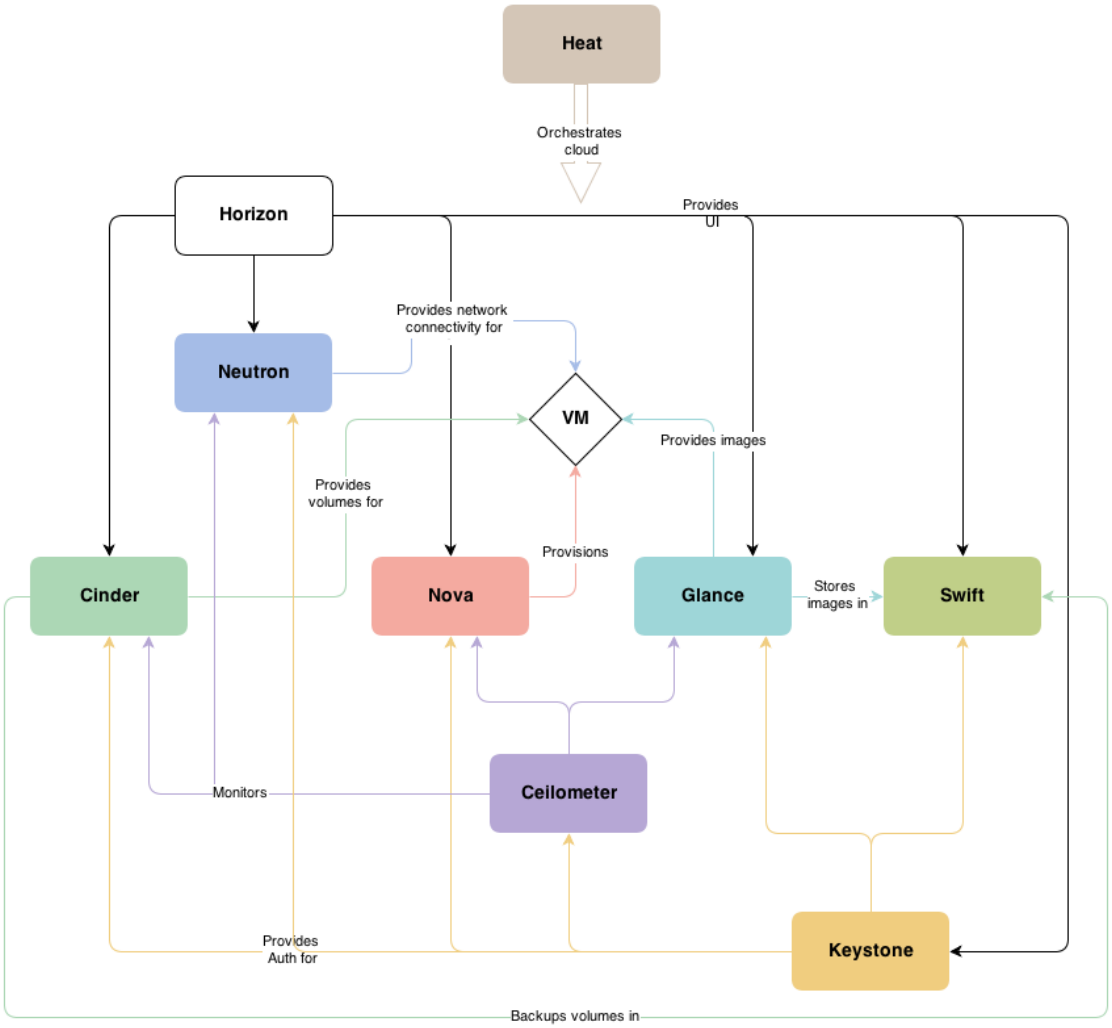

Fig. 7 Arquitectura modular OpenStack e interacciones entre sus componentes.

Tal y como se muestra en la Fig.7 anterior, los componentes [28] que constituyen la arquitectura modular de OpenStack son los siguientes:

- **Horizon**: Proporciona un portal de autoservicio basado en una WebGUI para interactuar con los servicios de OpenStack subyacentes, como el lanzamiento de una instancia, la asignación de direcciones IP y la configuración de los controles de acceso.
- **Nova**: Gestiona el ciclo de vida de las instancias de un proceso en un entorno OpenStack. Sus responsabilidades incluyen la creación, la programación y el borrado de máquinas virtuales. Similar al servicio EC2 de Amazon.
- **Neutron**: Proporciona conectividad de red entre diferentes dispositivos, los cuales están gestionados por otros servicios OpenStack. Además, proporciona una API (Application Programming Interface) para que los usuarios definan las redes y los archivos adjuntos en ellos. Además, posee una arquitectura conectable que soporta varios proveedores de redes y tecnologías populares.
- **Glance**: Almacena y proporciona un repositorio de imágenes de disco para las máquinas virtuales. El componente Nova hace uso de esta instancia durante el aprovisionamiento.
- **Heat**: Proporciona servicios de orquestación en la plataforma, organizando múltiples aplicaciones en la nube a través de un REST API nativa de OpenStack y una API Query compatible con CloudFormation.
- **Ceilometer**: Recoge las mediciones de la infraestructura y de las instancias, proporcionando un sistema de monitoreo para su posterior evaluación o para una posible facturación.
- **Cinder**: Administra el almacenamiento de bloques persistente (volúmenes de datos) que se pueden conectar a las instancias de las VMs.

Para el proyecto XIFI, los componentes que se utilizarán para el desarrollo de las nuevas funcionalidades del ABNO son Horizon, Nova y Neutron.

3.5 Controlador SDN local

En el proyecto XIFI, cada centro de datos se considera un dominio SDN independiente, de modo que cada dominio tiene su propio controlador, con responsabilidades de control sobre los recursos de red físicos y virtuales dentro del DC.

De acuerdo con la arquitectura de control en XIFI, el controlador de red XIFI ordena al controlador SDN local en las diferentes regiones o nodos instruir las reglas OpenFlow de reenvío en la infraestructura para construir la conectividad entre máquinas virtuales. Sólo las funcionalidades de empujar a las reglas OpenFlow de reenvío y descubrir la información de la topología serán utilizadas desde el controlador de cada nodo.

3.5.1 Controlador Floodlight

En un primer momento [29], el controlador SDN de referencia para la arquitectura XIFI fue Floodlight [30]. Se trata de un controlador de código abierto, desarrollado y mantenido por una comunidad abierta de desarrolladores, que funciona con switches físicos y virtuales.

Aunque al principio fue considerado como controlador local debido a su amplio uso y la extensa documentación acerca del mismo, posteriormente este controlador presentó algunos inconvenientes.

El inconveniente principal fue la falta de capacidad para el manejo de L3, dado que la solución XIFI tiene que induir capacidades L3, lo que hizo necesario sustituir este controlador por otra alternativa más potente. Se decidió así que el nuevo controlador local en el proyecto XIFI pasaría a ser Ryu.

3.5.2 Controlador Ryu

Actualmente como controlador SDN local se utiliza Ryu [31], el cual se define como un software definido en el marco de redes basado en componentes. Ryu es un controlador SDN de código abierto basado en Python que presenta capacidades L2 y L3. Además, ofrece componentes software con una API bien definida, que hace que sea fácil para los desarrolladores crear nuevas aplicaciones de gestión y control de la red.

Ryu es compatible con varios protocolos para la gestión de dispositivos de red, tales como OpenFlow, siendo totalmente compatible con las versiones 1.0, 1.2, 1.3 y 1.4 de OpenFlow. Todo el código [32] del controlador Ryu está disponible gratuitamente bajo la licencia Apache 2.0. Además, este controlador puede desempeñar el mismo papel que el controlador Floodlight y ofrecer mecanismos similares para la integración, con un API bien documentada [33].

3.5.2.1 Plugin Ryu

En las redes de conmutación de paquetes, como ocurre con la infraestructura XIFI, la QoS se puede definir como la capacidad de proporcionar diferentes prioridades para diferentes aplicaciones, usuarios o flujos de datos, o para garantizar un cierto nivel de rendimiento a un flujo de datos.

Actualmente, si bien es posible que un administrador de OpenStack puede hacer cumplir algunas de las políticas relacionadas con las máquinas virtuales, tales como asignar núcleos virtuales, cantidades de memoria RAM o disco, no existen mecanismos de código abierto para establecer y controlar la cantidad de tráfico generado por una máquina virtual, y para configurar una prioridad específica a algunos flujos que cruzan la red central.

Por lo tanto, es posible que un usuario inunde la red de la nube de la infraestructura con tráfico malicioso y sin ningún tipo de restricción, dañando así a otras aplicaciones que se ejecutan en la misma instalación. Por esta razón, se ha introducido también en la infraestructura XIFI algunos mecanismos [29] para hacer cumplir las políticas de calidad de servicio en OpenStack, entre ellos un plugin para Ryu.

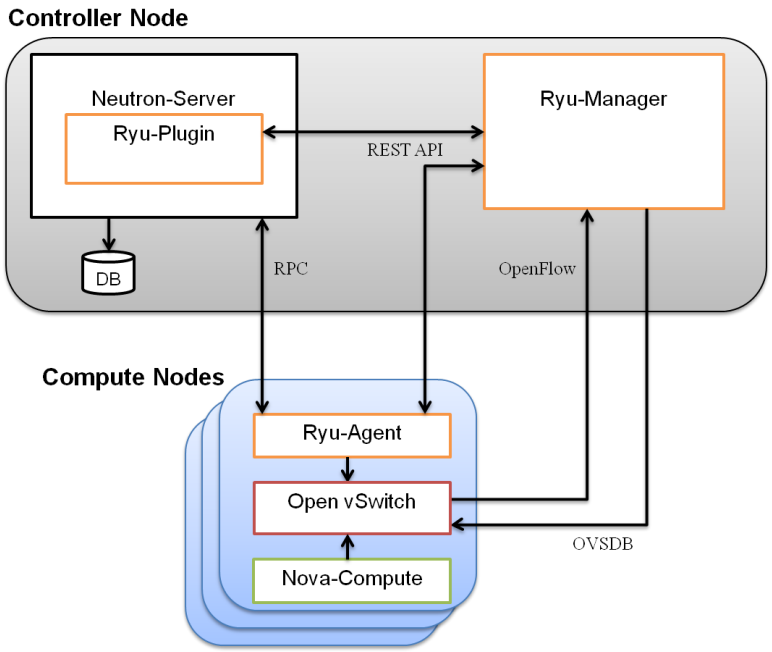

Fig. 8 Cooperación entre Neutron (OpenStack) y el plugin de Ryu.

Tal y como se muestra en la Fig. 8, el plugin Ryu actúa como regulador local SDN y configura el OVS (Open vSwitch) usando el protocolo correspondiente almacenado en la OVSDB (Open vSwitch DataBase).

4. Diseño e implementación de nuevas funcionalidades para la arquitectura ABNO

4.1 Workflow de creación de redes en DCs separados

A continuación se procede a explicar el escenario que se va utilizar como prueba de este proyecto, así como el procedimiento para la creación de las redes en el proyecto XIFI. Para ello se utilizará la enumeración de la siguiente figura.

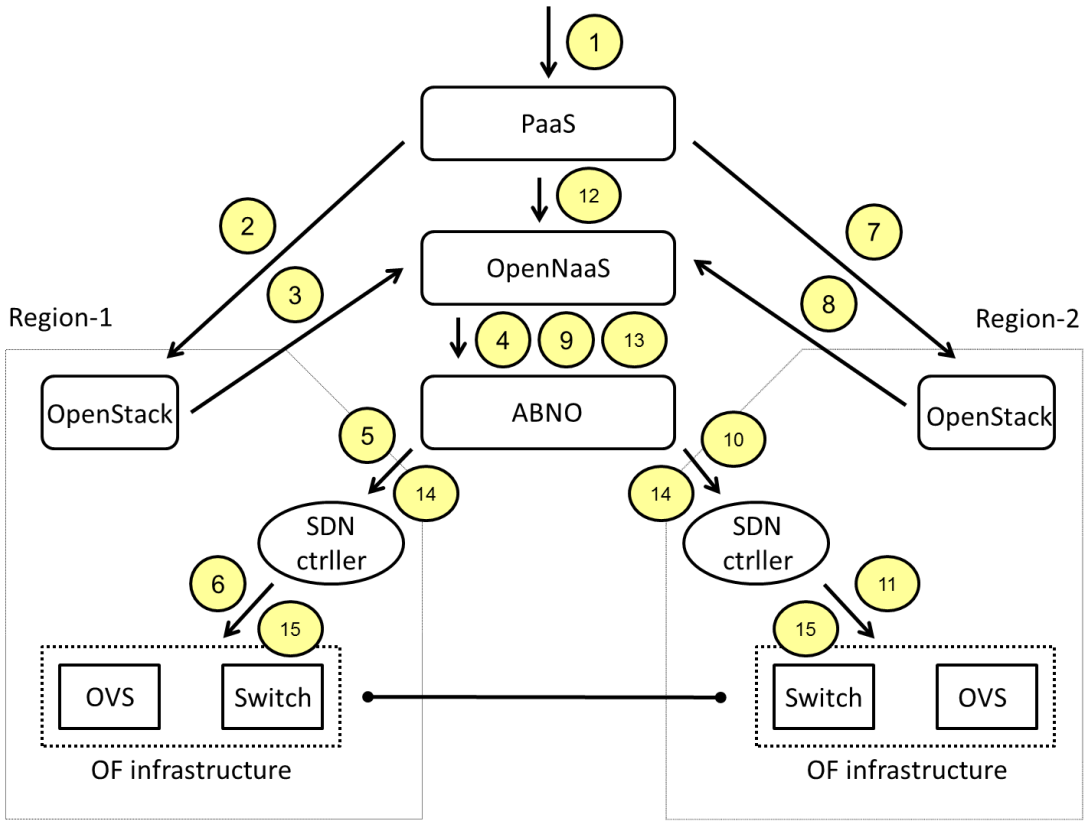

Fig. 9 Workflow de creación de redes en DCs separados.

1. A través del portal PaaS, un tenant o diente solicita las peticiones de despliegue de las VMs en diferentes regiones.

2. PaaS lanza la creación de red y subred en Region-1 donde se van a desplegar las VMs solicitadas por el tenant. El PaaS lanza la creación de una VM hacia la instancia OpenStack de Region-1.

3. Un plugin de Neutron en XIFI llama a OpenNaaS para la creación de reglas OpenFlow en el Open vSwitch.

4. OpenNaaS pasa indicación a ABNO de la VM recién creada.

5. ABNO procesa la información recibida y genera la petición de conectividad de acuerdo a la API del controlador RYU-SDN en Region-1.

6. El controlador RYU-SDN genera las reglas OpenFlow para configurar el Open vSwitch al que se conecta la VM.

7. PaaS lanza la creación de red y subred en Región-2 donde se van a desplegar las VMs solicitadas por el tenant. El PaaS lanza la creación de una VM hacia la instancia OpenStack de Region-2.

8. Un plugin de Neutron en XIFI llama a OpenNaaS para la creación de reglas OpenFlow en el Open vSwitch.

9. OpenNaaS pasa indicación a ABNO de la VM recién creada.

10. ABNO procesa la información recibida y genera la petición de conectividad de acuerdo a la API del controlador RYU-SDN en Region-2.

11. El controlador RYU-SDN genera las reglas OpenFlow para configurar el Open vSwitch al que se conecta la VM.

12. PaaS solicita a OpenNaaS la unión de las redes sobre las cuáles se han desplegado VMs en las distintas regiones.

13. OpenNaaS determina cuáles son las VMs involucradas en la unión de redes y genera las peticiones a ABNO.

14. ABNO instruye a los controladores RYU-SDN de cada región, empleando para ello la API correspondiente del controlador.

15. El controlador RYU-SDN de cada región genera las reglas OpenFlow para configurar los equipos OpenFlow bajo su control.

De esta forma, quedarían definidas las conexiones entre los DCs distribuidos e interconectadas las VMs correspondientes de cada región.

Si se profundiza en las interacciones que desempeñará el controlador ABNO con el resto de componentes del controlador de red XIFI, puede verificarse que ABNO deberá interactuar con OpenNaaS, con el controlador RYU-SDN y con OpenStack, como puede apreciarse en la siguiente Fig. 10.

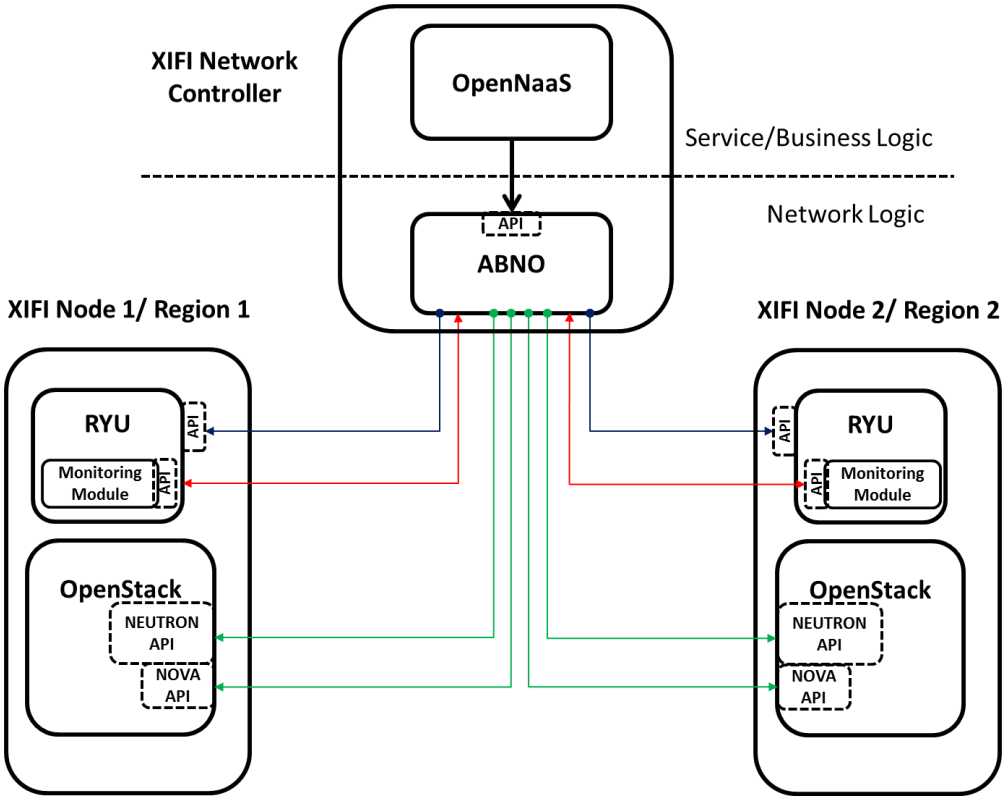

Fig. 10 Comunicación entre ABNO y el resto de componentes del controlador de red XIFI.

Todas estas interacciones entre ABNO con OpenNaaS, RYU y OpenStack serán detalladas en las siguientes secciones, así como las nuevas funcionalidades implementadas en el controlador ABNO para soportar estas comunicaciones con los nodos del proyecto XIFI.

4.2 Integración de OpenNaaS en ABNO

Como ya se ha detallado anteriormente, el módulo OpenNaaS es el encargado de la lógica de servicio, permitiendo abstraer la complejidad de la red subyacente con el fin de desplegar y operar fácilmente los servicios las redes de aprovisionamiento de acuerdo a las solicitudes, garantizándose así la calidad de servicio (QoS). Por lo tanto, el módulo OpenNaaS constituye el punto de entrada del controlador de red XIFI; recibe las solicitudes de servicio y convierte los requisitos en los parámetros comprensibles de red, de modo que el ABNO sea capaz de establecer la conexión E2E.

OpenNaaS supervisa el servicio de conectividad y mantiene el rendimiento de los servicios con una QoS garantizada. ABNO escucha las solicitudes E2E de conectividad de servicios de OpenNaaS por medio de la correspondiente interfaz y selecciona el flujo de trabajo adecuado a seguir, con el fin de conectar los recursos de red necesarios en los nodos XIFI e igualar la lógica de servicio. Como el módulo ABNO es el encargado de mantener la lógica del controlador de red XIFI, será el responsable de la orquestación de la red para el aprovisionamiento de las solicitudes de servicio que proporcionan la conectividad, invocando para ello a los componentes necesarios, en el orden correcto, de acuerdo a los flujos de trabajo específicos

La siguiente figura presenta la arquitectura modular del controlador de Red XIFI incluyendo OpenNaaS (color verde) y los componentes específicos del marco ABNO (color rojo) considerado para XIFI, ya que del conjunto de componentes funcionales propuestos en el RFC ABNO [20], sólo algunos de ellos son necesarios.

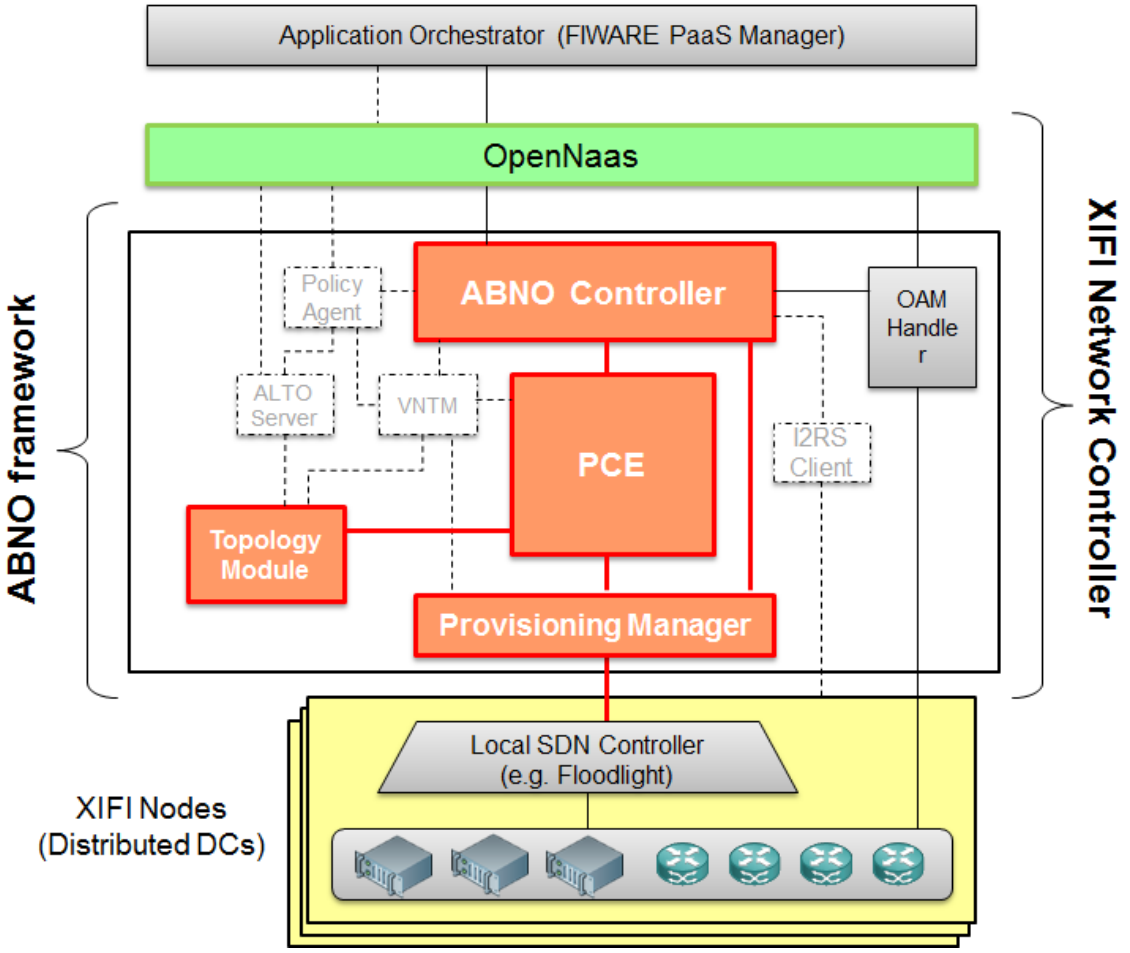

Fig. 11 Arquitectura modular del controlador de red XIFI.

La coordinación entre estas dos entidades resulta crucial para una implementación adecuada del servicio en general de la conectividad en XIFI. Esta comunicación entre OpenNaaS y ABNO es posible gracias a una REST (REpresentational State Trasfer Application) API (Application Programming Interface) capaz de recoger los parámetros necesarios para establecer una ruta E2E en la federación. Por otra parte, la API de OpenStack también es clave para una buena coordinación entre ABNO y OpenNaaS, y así lograr la correcta provisión de los servicios de red.

4.2.1 Nuevas funcionalidades implementadas en ABNO

En el orquestador ABNO han sido necesarias una serie de modificaciones para soportar la iteración con el componente OpenNaaS, el cual mantendrá la conciencia de servicio en la infraestructura XIFI.

Asimismo, OpenNaaS mantendrá la información relacionada con los recursos asociados a los inquilinos en cada DC, incluyendo identificadores que permiten el seguimiento del servicio de la conectividad. Esta información se corresponde por ejemplo con las direcciones MAC de las máquinas virtuales, los puertos donde las máquinas virtuales están asociadas a los correspondientes Open vSwitches y los identificadores locales de la red asignados por cada instancia local de OpenStack.

Al realizar OpenNaaS una petición de conectividad entre dos máquinas virtuales a ABNO, los principales parámetros que le proporciona OpenNaaS a ABNO son los siguientes:

- Región donde se encuentra desplegada la VM origen.
- Región donde se encuentra desplegada la VM destino.
- Dirección MAC de la VM origen.
- Dirección MAC de la VM destino.

Todos los parámetros necesarios por ABNO, así como la correspondiente API del código implementado se explican en detalle en el ANEXO C.

Con toda esta información recibida por parte de OpenNaaS, ABNO realiza las siguientes acciones:

1) Calcula si existe un posible camino entre las dos máquinas virtuales desplegadas en las correspondientes regiones. La información correspondiente a cada región está almacenada en ficheros xml con el mismo nombre de la región, los cuales se actualizan y están disponibles para el controlador ABNO. El formato y la estructura de dichos ficheros se describe con detalle en el ANEXO A.2.

2) En caso de existir un camino entre las dos VMs, ABNO intenta configurar los switches, instruyendo para ello las correspondientes reglas OpenFlow a los controladores Ryu de cada región. Este paso, así como la integración con Ryu, se detallan en el punto 4.4 de este trabajo.

3) Si ha sido posible instruir las reglas a los controladores Ryu y configurar los switches, ABNO devolverá a OpenNaaS <200 OK > en formato "json", para informar de que ha sido posible establecer la conexión entre las dos máquinas virtuales. En caso de que alguno de los pasos anteriores hubiera fallado ABNO responderá a OpenNaaS, también en formato "json" con los siguientes códigos y mensajes de error:

   - **< 400 Bad Request >** : Client Params.

     Si el problema se ha producido por un error en los parámetros de la solicitud proporcionada por OpenNaaS.

   - **< 500 Internal Server Error >** : No Open vSwitch available. Not posible to provide a path.

     Si el problema se ha producido por un error interno del servidor, ya sea que no están disponible las VMs, que no se hayan podido instruir las reglas Ryu o que exista algún error en los ficheros de configuración de las correspondientes regiones donde se debían desplegar las VMs.

4.3 Integración de OpenStack en ABNO

4.3.1 Instalación, configuración y solución de errores OpenStack

Cada nodo de la infraestructura XIFI debe tener instalado y configurado correctamente el software de computación en la nube OpenStack. Más concretamente, se ha utilizado la versión de OpenStack para desarrolladores (DevStack) [34] con el objetivo de poder soportar todas la funcionalidades necesarias de Ryu, así como el plugin de Ryu para Neutron y asegurar la QoS. Es muy importante que DevStack esté configurado correctamente ya que supondrá la base para la instalación y configuración del resto de componentes.

Antes de proceder con la instalación de DevStack, es importante que los nodos[1] XIFI tengan instalada una versión de Python 2.7. En caso de no ser así, solo será necesario ejecutar desde la consola del nodo XIFI lo siguiente:

```
$ wget http://python-distribute.org/distribute_setup.py
$ sudo python distribute_setup.py
```

A continuación ya se puede proceder a la instalación de DevStack[2], tarea que puede resultar bastante compleja en un primer momento, a pesar de que su uso posterior resulta bastante intuitivo. Los pasos para su instalación, desde la consola de los nodos XIFI, se detallan a continuación.

1) Descargar DevStack mediante el repositorio git correspondiente.

```
$ git clone https://github.com/openstack-dev/devstack.git
  -b stable/icehouse
```

2) Desplazarse hasta el directorio donde se ha descargado DevStack.

```
$ cd devstack
```

3) Crear un fichero con el nombre "localrc", el cual contendrá la información sobre las correspondientes IPs de nuestro entorno y los servicios que deben activarse en OpenStack. Este fichero de configuración se detalla en el ANEXO A.1.

4) Iniciar la instalación de DevStack.

```
$ ./stack.sh
```

Durante la instalación de DevStack pueden ocurrir algunos problemas, los cuales han sido solucionados de la siguiente forma:

[1] Los servidores que se han utilizado para la instalación de DevStack, y simular los nodos de la infraestructura XIFI, tienen una distribución Ubuntu 14.04 LTS con un kernel 3.13.0-24-generic x86-64. Además deben tener instalada una versión de Java igual o superior a 1.6.

[2] La versión de OpenStack, DevStack, que se ha utilizado ha sido "IceHouse".

a) Error en las urls del fichero "devstack/stackrc", lo cual se puede solucionar ejecutando el siguiente comando desde el directorio devstak:

```
$ sed -i 's/git:/https:/g' stackrc
```

b) Error en el paquete "python-amqp". Para solucionarlo solo es necesario ejecutar el siguiente comando:

```
$ install python-amqp 1.4.5
```

c) Error en el paquete "python-lxml". Para solucionarlo basta con ejecutar:

```
$ sudo apt-get libxml2-dev libxslt1-dev python-dev
```

d) Error en la instalación que realiza DevStack de mysql 5.5. Se soluciona ejecutando los siguientes comandos:

```
$ sudo apt-get purge mysql-client-core-5.6
$ sudo apt-get autoremove
$ sudo apt-get autoclean
$ sudo apt-get install mysql-client-core-5.5
$ sudo apt-get autoremove
```

e) Error en algún fichero de los directorios:
   - /opt/stack
   - /usr/local/lib/python2.7/dist-packages

   Este error suele suceder cuando hemos tenido algún fallo en la instalación de DevStack y hemos intentado reinstalarlo. Para solucionarlo basta con limpiar el directorio afectado, los comandos necesarios son:

```
$ rm -rf /opt/stack
$ rm -rf /usr/local/lib/python2.7/dist-packages/*
```

Es importante destacar, que después de corregir cualquier error durante la instalación de DevStack, es necesario detener la instalación antigua que existía y limpiar los ficheros que

se han ido creando antes de volver a ejecutar de nuevo DevStack. Para ello es necesario ejecutar los siguientes comandos desde el directorio devstack:

```
$ ./unstack.sh
$ ./clean.sh
```

Si durante la instalación de DevStack surge algún error diferente a los explicados anteriormente, puede visualizarse el error de una manera más detalla ejecutando lo siguiente[3]:

```
$ screen -r stack
```

Una vez instalado DevStack y solucionados los posibles errores, es necesario configurar algunas características para poder usar MDVPN (Multi-Domain Virtual Private Network) con OpenStack.

❖ Configuración del controlador Open vSwitch[4].

- Fijar la IP del controlador Open vSwitch. Si el controlador está en la misma máquina que el Open vSwitch, la IP se corresponderá con: 127.0.0.1, siendo distinta en otro caso.

```
$ sudo ovs-vsctl set-controller br-int tcp:127.0.0.1:6633
```

❖ Configuración del puerto br-ex del Open vSwitch.

- Añadir la interfaz física (por ejemplo eth0) donde MDVPN es alcanzable, a puerto br-ex del switch:

```
$ sudo ovs-vsctl add-port br-ex eth0
```

- Eliminar la IP de esta interfaz física y restaurar (si es necesario) las reglas de las rutas:

[3] Si se presiona Ctrl+A+N se puede ir pasando de pantalla y visualizar todos los servicios que se encuentran en ejecución.
[4] Todos los comandos correspondientes a Open VSwitch pueden encontrarse en el ANEXO B.1.

```
$ sudo ifconfig eth0 0
```

Todos los comandos correspondientes a la creación de VMs, creación de redes y consulta de información OpenStack [28] se encuentran detallados en el ANEXO B.2.

4.3.2 Nuevas funcionalidades implementadas en ABNO

La interacción entre OpenStack y el controlador ABNO es imprescindible para obtener la información necesaria acerca de las VMs y sus puertos asociados.

Para poder realizar la integración entre estos dos componentes, han sido necesarias una serie de pruebas previas a la implementación del nuevo código. Estas pruebas se basan en la creación de unos scripts, lo cuales son capaces de obtener los puertos específicos del Open vSwitch en donde está conectada cada VM de cada instancia de OpenStack. Los comandos que ejecutan estos scripts para obtener los puertos deseados son los siguientes:

- El primer paso es conocer y obtener el identificador de la VM a partir del nombre de la VM.

```
id_VM=`nova list | grep $1 | grep -o '\<[0-9a-z\.-]*\>'
       | head -1`
```

- Con este identificador de la VM, puede ser descubierto el identificador del puerto en el que la VM está conectado al Open vSwitch.

```
id_port_VM=`neutron port-list --device_id "id_VM" |
            grep -o '\<[0-9a-z\]*\>' | head -3 |
            grep -o '\<........\>'`
```

- A partir del identificador del puerto del Open vSwitch al que está conectado la VM, se obtiene el identificador "qvo" asociado a esa VM.

```
qvo_VM=`ifconfig | grep "$id_port_VM" | grep qvo |
        grep -o '\<[0-9a-z\.-]*\>' | head -1`
```

- Finalmente, con este identificador del tipo "qvo" de la VM y consultando al Open vSwitch, obtenemos el puerto buscado.

```
ofport_VM=`sudo ovs-vsctl list Interface "$qvo_VM" |
            grep ofport | grep -o '\<[0-9]\>'`
```

Todo este proceso de obtención de los puertos del Open vSwitch asociados a las VMs se ha integrado dentro del software ABNO, afectando en especial al módulo ABNO Controller.

Dentro del software del módulo ABNO Controller, existe un workflow específico encargado de esta tarea, el cual lanza una conexión SSH (Secure Shell) y ejecuta el siguiente comando en la máquina remota con el fin de obtener el identificador del puerto deseado.

```
$ sudo ovs-vsctl list Interface "+param+"| grep ofport |
  head -1 | tr -d [ofport] | tr -d [:punct:] | sed
  's/[[:space:]]*$//' | sed 's/^[[:space:]]*//'
```

El parámetro "param" hace referencia al parámetro "source_interface"[5] de la VM, el cual ha sido proporcionado previamente en la solicitud de OpenNaaS al controlador ABNO. El resultado de esta ejecución es el número del puerto del Open vSwitch deseado.

En el futuro, el acceso para la obtención del identificador de puertos se realizará mediante un token de autenticación obtenido a través del componente Keystone de OpenStak, con el fin de evitar conexiones SSH directas y los posibles problemas de seguridad asociados.

4.4 Integración de Ryu en ABNO

4.4.1 Instalación, configuración y solución de errores Ryu

Cada nodo de la infraestructura XIFI además de tener el software de computación en la nube OpenStack, debe tener instalado y configurado correctamente Ryu, con el fin de poder controlar cada nodo XIFI por ABNO, y por lo tanto por OpenNaaS.

Como ya se ha explicado, como cada centro de datos se considera un dominio SDN independiente, por lo que cada DC debe tener su propio controlador Ryu, con responsabilidades de control sobre los recursos de red físicos y virtuales. Solo se utilizarán desde este controlador las funcionalidades de empujar las reglas de reenvío y descubrir la información de la topología.

[5] Véase ANEXO C con la API detallada del controlador ABNO.

Antes de proceder con la instalación de Ryu, es necesario tener instalada una versión de Python 2.7 o superior. Una vez instalado Python, existen dos alternativas para instalar el software Ryu[6].

La primera de ellas es a través del comando "pip".

```
$ sudo pip install ryu
```

La segunda opción es a través del repositorio git, ejecutando el siguiente comando en el directorio deseado.

```
$ git clone https://github.com/osrg/ryu.git
```

Para comprobar que la instalación ha ido correctamente basta con ejecutar lo siguiente:

```
$ ryu-manager --verbose
```

Durante la instalación de Ryu se ha encontrado un problema en el fichero:

- */ryu/ryu/lib/packet/lldp.py

El problema estaba asociado al recorrido de un bude fuera de rango, por lo que para solucionar este inconveniente, se ha propuesto una solución modificando dicho fichero cambiando la forma en recorrer el bude y añadiendo un contador al mismo. Los cambios que se han realizado en el fichero "lldpy.py" puede verse detallados al completo en el ANEXO A.3.

Todos los comandos correspondientes para instruir o eliminar reglas en los switches a través de Ryu SDN [33] se encuentran detallados en el ANEXO B.3.

[6] La versión de Ryu que se ha utilizado ha sido Ryu 3.13

4.4.2 Pruebas con mininet

Antes de integrar el componente Ryu con el controlador ABNO, se han realizado una serie de pruebas con mininet.

Mininet [35] es un emulador de red que crea una red virtual realista, haciendo correr para ello en segundos un núdeo verdadero, switches y código de la aplicación, en una sola máquina, como es el caso de una VM.

Para instalar mininet basta con ejecutar en la consola:

```
$ sudo apt-get install mininet
```

Para comprobar que la instalación ha ido correctamente, es necesario ejecutar el siguiente comando, el cual comprobará la conectividad.

```
$ sudo mn --test pingall
```

Cuando ya ha sido instalado mininet, para crear una topología básica con 3 switches será necesario ejecutar:

```
$ sudo mn --topo linear,3 --mac
  --controller=remote,ip=127.0.0.1 --switch ovsk
```

A continuación, ya se puede interactuar con los switches a través de la consola que se mostrará. Para añadir las reglas [36] entre dos de los switches creados, y así simular el comportamiento que tendría Ryu, basta con ejecutar lo siguiente:

```
mininet> s1 ovs-ofctl add-flow "s1" priority=1,ip,
         nw_dst=<host2_ip>,actions=mod_dl_dst:<host2_mac>,
         output=2
```

La regla creada indica que todo el tráfico que entra el switch, se extraiga por el puerto 2 del mismo.

4.4.3 Nuevas funcionalidades implementadas en ABNO

Una vez entendido el funcionamiento de Ryu con la simulación de la topología a través de mininet, se procede la integración del controlador Ryu SDN con el software ABNO, afectando en especial al módulo ABNO Controller, más concretamente al componente Provisioning Manager.

El controlador ABNO, gracias a la información topológica almacenada en los ficheros correspondientes a cada región descritos en el ANEXO A.2, es capaz de instruir las reglas a los controladores locales Ryu. A partir de los ficheros de las regiones, ABNO obtiene las IPs de los controladores de cada nodo, mientras que los puertos y los datapaths de los switches necesarios para instruir las reglas Ryu se obtienen a partir los puertos y las direcciones MAC que se encuentran en la solicitud realizada por OpenNaaS.

La forma de introducir una regla Ryu es bastante fácil y sigue la siguiente estructura:

```
$ curl -X POST -d '{"cookie": "1","dpid": "$DPID_PARAM",
  "match": {"in_port":"10"}, "actions": [{"type": "OUTPUT",
  "port": "33"}]}'
   http://$CONTROLLER_IP:8080/stats/flowentry/add
```

El ejemplo de regla anterior muestra como todo el flujo de tráfico que entra por el puerto 10 (`{"in_port":"10"}`), es sacado por el puerto 33 (`{"type": "OUTPUT","port": "33"}`); donde el parámetro `"$DPID_PARAM"`, se corresponde con el valor decimal del dapathid del Open vSwitch donde se desea establecer la regla y `"CONTROLLER_IP"` con la IP del nodo donde se encuentra el controlador Ryu. De manera que uno de los puertos especificados en la regla se corresponderá con el puerto de entrada de flujo de tráfico al Open vSwitch, y el otro puerto el de salida, se corresponderá al puerto que esté conectado al Túnel GRE que se detallará en la siguiente sección. También ocurrirá lo contrario, que el puerto de entrada al Open vSwitch sea el que esté conectado al Túnel GRE y el otro puerto sea el de salida del flujo de tráfico, dependiendo de la localización del Open vSwitch.

Si se desea establecer la conectividad entre los puertos A y B de un switch será necesario instruir dos reglas diferentes, una de ellas para especificar que todo el tráfico que entre por el puerto A se extraiga por el puerto B, y otra con la operación contraria, donde el tráfico introducido por el puerto B salga por el puerto A.

De esta manera ABNO va estableciendo la conectividad E2E solicitada por OpenNaaS. Para ver las reglas configuradas en cada Open vSwitch puede ejecutarse el siguiente comando:

```
$ sudo ovs-ofctl dump-flows br-int
```

El resto de comandos correspondientes a los Open vSwitches puede encontrarse en el ANEXO B.1.

4.5 Integración Túneles GRE con ABNO

La conectividad de la red troncal de la federación XIFI se basa en el transporte MDVPN ofrecido por la comunidad GÉANT. Esta conectividad entre los diferentes nodos XIFI se realizará utilizando Túneles GRE.

4.5.1 Instalación, configuración y solución de errores Túneles GRE

El proceso para la instalación y configuración de los túneles GRE sigue los pasos descritos a continuación, los cuales deberán ser repetidos en ambos nodos, en los puertos "br-ex" y "br-int" creados en los Open vSwitches como resultado de la instalación OpenStack.

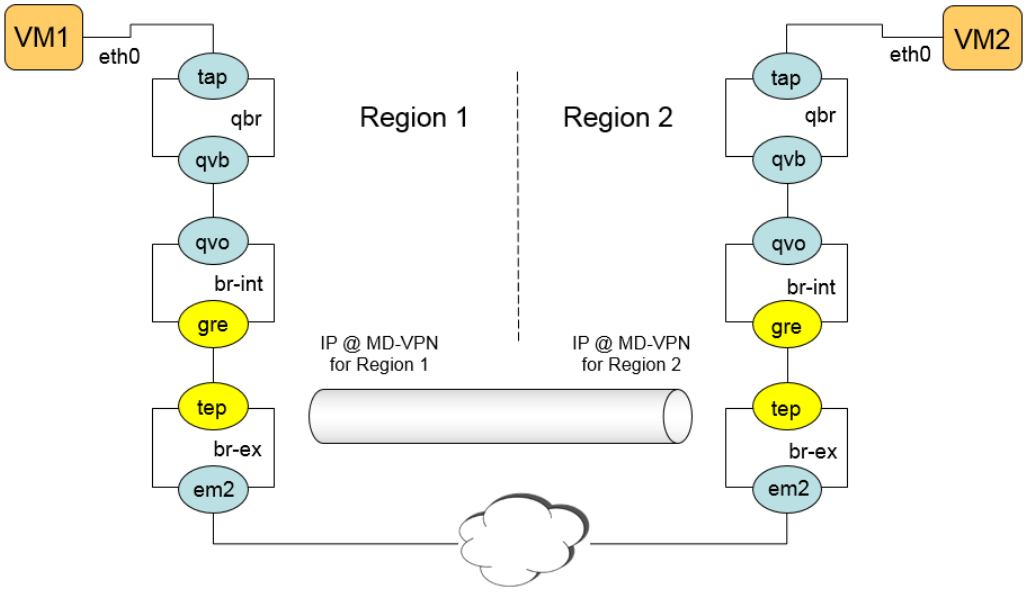

Fig. 12 Detalle Túneles GRE.

❖ **Configuración de los puertos finales "br-ex".**

El puerto final del túnel GRE (tep) actuará como punto final para la conexión del Túnel GRE en cada extremo. A este puerto tep se le asignará una dirección IP del MDVPN, con el fin de permitir la conectividad con otros centros de datos en la federación.

Para crear la interfaz interna, es necesario ejecutar el siguiente comando[7]:

```
$ sudo ovs-vsctl add-port br-ex tep0 -- set interface
  tep0 type=internal
```

Una vez que la interfaz interna ha sido definida, es necesario asignarle una dirección IP[8] del MDVPN usando para ello el comando que se prefiera entre "ifconfig" o "ip":

```
$ sudo ifconfig tep0 $IP_ASSIGNED netmask 255.255.255.0
```

En este punto, una vez que se han establecido los puntos finales de los Túneles GRE en cada servidor, es recomendable probar la conectividad entre los extremos del túnel por medio del comando "ping" o una herramienta similar.

❖ **Establecimiento del Túnel GRE.**

Para establecer el Túnel GRE es necesario agregar además una interfaz GRE al puerto "br-int" en cada servidor:

```
$ ovs-vsctl add-port br-int gre0 -- set interface gre0
  type=gre options:remote_ip=<$OTHER_GRE_TUNNEL_ENDPOINT>
```

Donde `"$OTHER_GRE_TUNNEL_ENDPOINT"` se refiere a la IP del otro servidor definida en el paso anterior como `"$IP_ASSIGNED"`.

Una vez que los Túneles GRE han sido desplegados en los centros de datos independientes, el tráfico que se intercambia entre las máquinas virtuales que residen en cada centro de datos se remitirá a través de ellos.

[7] Sería recomendable realizar previamente una planificación los nombres de identificadores que se van a utilizar para definir los túneles.

[8] Los rangos IP separados del MDVPN deben ser identificados. La IP a asignar deberá ser diferente a la asignada en la tarjeta de interfaz de red física del servidor.

4.5.2 Nuevas funcionalidades implementadas en ABNO

Para poder realizar la integración entre los Túneles GRE y ABNO, y así poder establecer la conectividad E2E en la federación XIFI, han sido necesarias una serie de pruebas previas a la implementación del nuevo código. Estas pruebas en la creación de unos scripts, lo cuales son capaces de obtener los puertos específicos de los Túneles GRE definidos para la conexión de extremo a extremo a través de MDVPN. El comando que ejecutan estos scripts para obtener los puertos deseados es el siguiente:

```
ofport_GRE=`sudo ovs-vsctl list Interface $NAME_TUNNEL_GRE
           | grep ofport | grep -o '\<[0-9]\>'`
```

Este comando devuelve el puerto buscado del túnel GRE a partir del parámetro "$NAME_TUNNEL_GRE", el cual se corresponde con el nombre del Túnel GRE del cual se desea conocer el puerto.

Todo el proceso de obtención de los puertos de los Túneles GRE, se ha integrado dentro del software ABNO, afectando en especial al módulo ABNO Controller.

Dentro del software del módulo ABNO Controller, existe un workflow específico encargado de esta tarea, el cual lanza una conexión SSH. Una vez dentro de la máquina remota, se ejecuta el siguiente comando con el fin de obtener el identificador del puerto asociado al Túnel GRE deseado.

```
$ sudo ovs-vsctl list Interface "+param+"| grep ofport |
  head -1 | tr -d [ofport] | tr -d [:punct:] | sed
  's/[[:space:]]*$//' | sed 's/^[[:space:]]*//'
```

El parámetro "param" se corresponde con el nombre del Túnel GRE que se encuentra definido en el fichero de configuración correspondiente al nodo XIFI del cual se desea conocer el puerto. El resultado de esta ejecución es el número del puerto del Túnel GRE deseado.

4.6 Validación de las nuevas funcionalidades implementadas en la arquitectura ABNO

Con el fin de emular un verdadero ambiente de conectividad, a continuación se detalla la simulación de la infraestructura XIFI y las pruebas que se han realizado.

4.6.1 Configuración de la topología física utilizada

Para la validación de las nuevas funcionalidades en el controlador/orquestador ABNO se ha desplegado un escenario simulando 3 nodos de la infraestructura XIFI. Uno de ellos se localiza en un servidor de Madrid (RedIris), el segundo nodo se sitúa en Trento, Italia, (Create-Net) y el último en Irlanda (TSSG).

Con el objetivo de tener una visión más clara de cómo va a ser esta topología, así como los datos sobre puertos, direcciones MAC, direcciones IP o identificadores datapath necesarios para la construcción de las conexiones E2E entre los diferentes nodos XIFI, se han realizado los siguientes esquemas agrupando toda esta información.

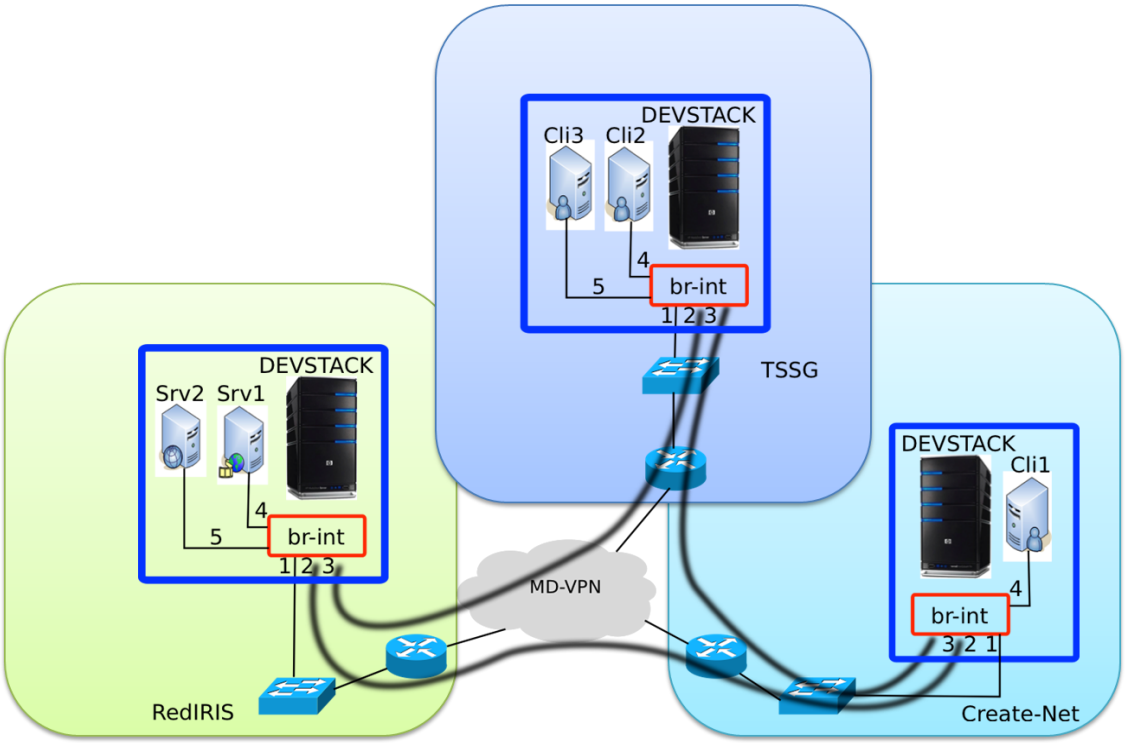

Fig. 13 Topología del caso de uso.

Un centro de datos está constituido por un switch OpenFlow habilitado y un servidor. Los switches[9] constituyen los puntos de demarcación. Las conexiones entre los switches son caminos de superposición. Para las pruebas realizadas se ha seleccionado una conexión VPN de capa 2 entre los centros de datos.

[9] Los switches utilizados son 5406zl de HP, soporte OpenFlow 1.0.

Los tres servidores se utilizan para albergar, tanto a los componentes de software que se describen más adelante, como a las máquinas virtuales instanciadas por los usuarios finales. Los servidores están unidos directamente a los switches físicos a través de enlaces punto a punto.

Node Name	MD-VPN IP	MDVPN POOL
Create-Net	10.0.35.1	10.0.35.1 - 10.0.35.253
TSSG	10.0.15.109	10.0.15.90-10.0.15.99
RedIRIS	10.0.78.2	10.0.78.3 - 10.0.78.254

Tabla 1 Información detallada de los nodos físicos.

ID	Location	MD-VPN IP	Int. IP	MAC
Client-1	Create-Net	10.0.35.4	10.0.0.3	fa:16:3e:ec:e2:cb
Client-2	TSSG	10.0.15.91	10.0.0.4	fa:16:3e:4c:5c:b0
Client-3	TSSG	10.0.15.92	10.0.0.5	fa:16:3e:e1:67:cf
Srv1	RedIRIS	10.0.78.4	10.0.0.6	fa:16:3e:2a:79:b0
Srv2	RedIRIS	10.0.78.5	10.0.0.7	fa:16:3e:f6:a6:7a

Tabla 2 Información detallada de las máquinas virtuales desplegadas.

Los componentes software necesarios descritos anteriormente se implementan a nivel local en cada uno de los centros de datos para gestionar y controlar los recursos de computación. Sólo el controlador de red XIFI, actuando como orquestador de la red, es centralizado, ya que tiene toda la visión de la federación.

Un MPLS (Generalized Multiprotocol Label Switching) de capa 2, de servicio tradicional VPN, ha sido el encargado de proporcionar la conexión inter-DC. VLAN dedicadas han sido definidas para cada centro de datos con el fin de distinguir las diferentes conexiones, estando configuradas en los routers donde los switches están conectados.

Con el objetivo de poder establecer la conexión entre los 3 nodos de la simulación, es necesario el despliegue de Túneles GRE entre los nodos como muestra la siguiente Fig. 14. La configuración de estos nodos sigue el esquema descrito en el punto 4.5.1.

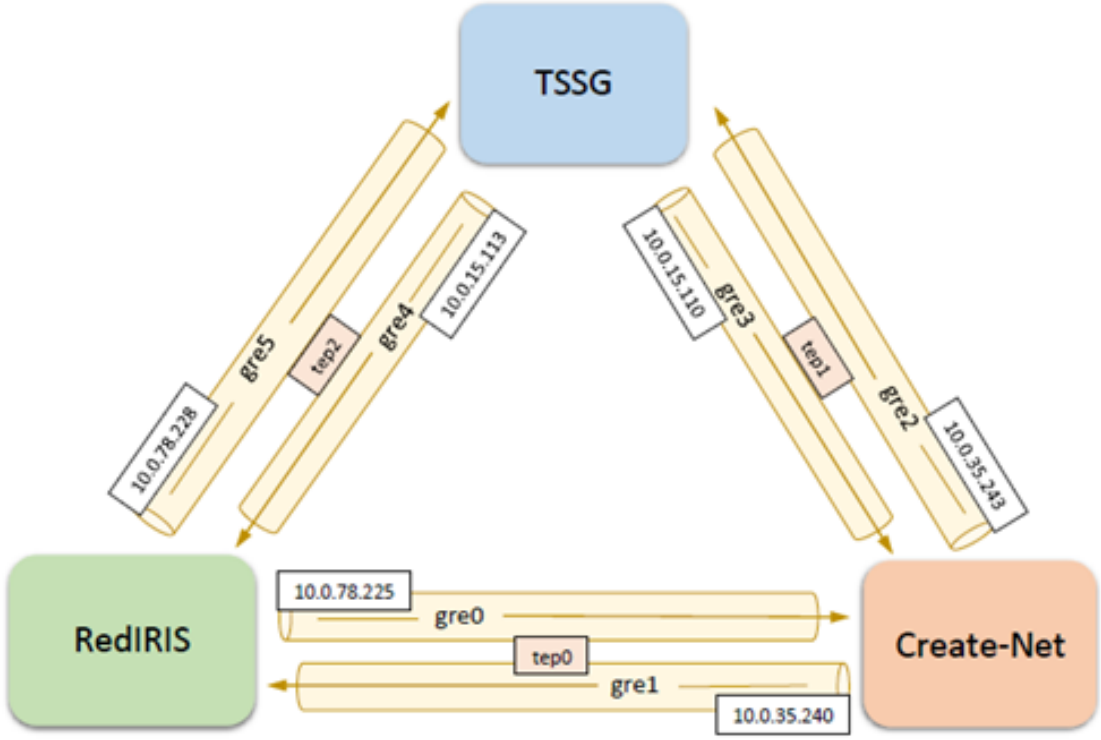

Fig. 14 Despliegue Túneles GRE.

4.6.2 Interacciones entre los componentes de la arquitectura ABNO durante las pruebas de conectividad realizadas

Dado que el orquestador ABNO es el encargado de establecer la conectividad extremo a extremo en la federación XIFI, para ello ABNO instruye, como hemos visto, a los controladores Ryu SDN locales de cada DC a poblar las reglas OpenFlow de reenvío.

La siguiente Fig. 15 muestra paso a paso cómo ha actuado ABNO ante las peticiones de recibidas por OpenNaaS durante las pruebas realizadas para simular la conectividad E2E entre dos nodos. Para ello, se muestran todas las interacciones entre los principales módulos del controlador ABNO, desde que se recibe la solicitud de OpenNaaS, se procesa a través de los componentes de la arquitectura ABNO y se instruyen las correspondientes reglas Ryu a los nodos de la infraestructura XIFI.

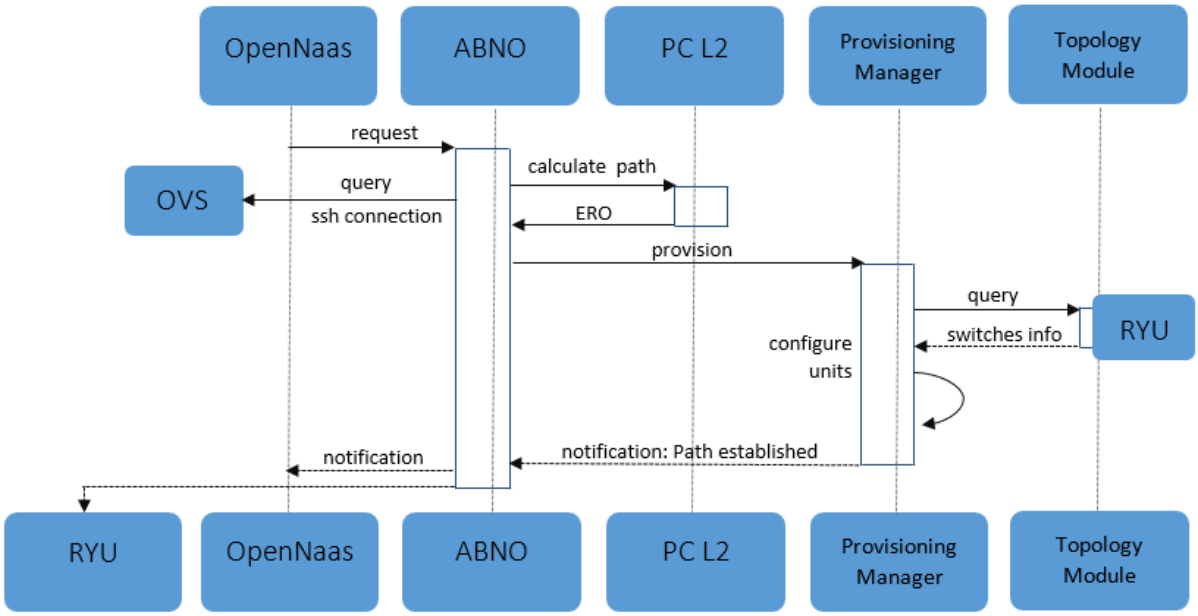

Fig. 15 Interacciones entre los diferentes módulos de la arquitectura ABNO.

A continuación, se detallan cada uno de los pasos que ha realizado ABNO durante las pruebas para realizar una provisión punto a punto entre 2 nodos de la simulación:

- OpenNaaS envía la solicitud de la petición de conectividad al controlador ABNO induyendo las regiones y las direcciones MAC de origen y destino, así como los identificadores los puertos de las VMs a través de los cuales los hosts están conectados al Open vSwitch.

  Esta interfaz de comunicación entre OpenNaaS y ABNO es una interfaz del tipo JSON.

- Cuando el controlador ABNO recibe la solicitud, éste obtiene la información requerida a través de los archivos de configuración gracias a la información que contenía la petición de OpenNaaS. Con toda esta información, ABNO establece una conexión SSH con el fin de obtener los puertos que necesita de los OVS.

- ABNO pregunta al PCE de capa 0 por un camino entre los dos nodos. El protocolo PCEP (Path Computation Element communication Protocol) es utilizado en esta interfaz.
- El PCE responde al controlador ABNO con una respuesta de tipo PCEP.
- El controlador ABNO crea una paquete PCEP inicial a partir de la respuesta del PCE y la envía al Provisioning Manager.
- El Provisioning Manager con al Topology Module para obtener la descripción de cada nodo.
- Dependiendo de la tecnología y del modo de configuración, selecciona un protocolo diferente para completar la solicitud.
- Se instruyen las correspondientes reglas al controlador Ryu SDN local y se establece la conexión entre las VMs.
- Una vez que la ruta se ha establecido, se le notifica al ABNO Controller, mediante la interfaz PCEP.
- Del mismo modo, el ABNO Controller anuncia a OpenNaaS si todo está correcto, o ha habido algún un problema, mediante la interfaz JSON.

    - Las respuestas que se envían a OpenNaaS siguen estas estructuras:

        i. Code: **400. SC_BAD_REQUEST**. En caso de que OpenNaaS haya enviado unos parámetros incorrectos a ABNO.

        ii. Code: **500. SC_INTERNAL_SERVER_ERROR**. En caso de que ABNO no fuese capaz de establecer la ruta o de instruir las correspondientes reglas Ryu. Un ejemplo podría ser: “No Open VSwitch available. Not possible to provide a path".

        iii. Code: **200. SC_OK**. E caso de que todo hay ido bien.

4.6.3 Validación

Tal y como se ha comentado anteriormente durante la explicación del escenario de la simulación XIFI, se han realizado pruebas de conectividad E2E entre 2 nodos mediante posibles peticiones OpenNaaS al controlador ABNO.

- Ejemplo de petición a ABNO de conectividad de una VM de Madrid con una VM de Trento:

```
$ curl "localhost:4445?source_region=Madrid&dest_region=
  Trento&source_mac= fa:16:3e:2a:79:b0&dest_mac=
  fa:16:3e:ec:e2:cb&source_interface=20&
  destination_interface=20&operation=WLAN_PATH_PROVISIONING&
  Operation_Type=XifiWF&ID_Operation=1234"
```

- Ejemplo de petición a ABNO de conectividad de dos VMs de Madrid con dos VMs de Irlanda:

```
$ curl "localhost:4445?source_region=Madrid&dest_region=
  Irlanda&source_mac= {fa:16:3e:2a:79:b0, fa:16:3e:f6:a6:7a}
  &dest_mac={fa:16:3e:4c:5c:b0, fa:16:3e:e1:67:cf}
  &source_interface=20&destination_interface=20&operation=
  WLAN_PATH_PROVISIONING&Operation_Type=XifiWF&
  ID_Operation=1234"
```

Se puede ver un ejemplo de la interacción entre ABNO y OpenNaaS en la captura de Wireshark de la Fig.16, donde se aprecia el intercambio de mensajes que realizan.

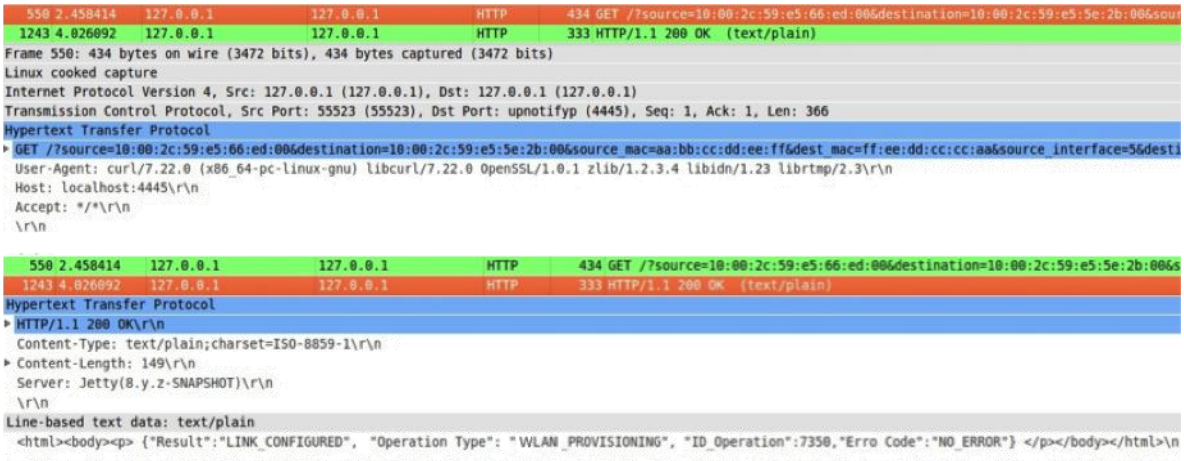

Fig. 16 Intercambio de mensajes para proporcionar conectividad E2E.

5. Conclusiones y futuras líneas de trabajo

5.1 Conclusiones

En este trabajo final de grado se ha contribuido a la interconexión de centros de datos distribuidos geográficamente, configurando para ello los componentes software necesarios e integrando nuevas funcionalidades en el controlador ABNO.

Para poder realizar el despliegue del correspondiente escenario de la infraestructura XIFI ha sido necesario realizar un profundo estudio de los componentes software necesarios, OpenStack, Ryu y Túneles GRE, con el fin de diseñar cómo establecer correctamente la conectividad entre los nodos de la infraestructura. Una vez entendidos, se ha procedido a su configuración en las diferentes regiones, así como la corrección de los errores que se han ido encontrando.

La segunda contribución de este trabajo ha sido el desarrollo, implementación y validación de las nuevas funcionalidades en el componente ABNO, capaz de orquestar las diferentes máquinas virtuales que se encuentren desplegadas en cada nodo XIFI, de instruir las correspondientes reglas Ryu y así establecer la conectividad E2E.

Además, también se ha contribuido a integración de la arquitectura ABNO con OpenNaaS, este último componente es el encargado de enviar las solicitudes de petición al controlador ABNO. Junto la información proporcionada en estas solicitudes y los ficheros de configuración, el controlador ABNO es capaz de establecer la conectividad en la infraestructura XIFI.

La solución presentada en este trabajo final de grado ha sido probada en una federación con infraestructuras multidominio.

5.2 Trabajo Futuro

Existen varias líneas de trabajo futuro. La primera de ellas consistiría en prescindir de las conexiones directas por SSH y así evitar los posibles problemas de seguridad asociados. Para ello, la obtención del identificador de puertos se realizaría mediante un token de autenticación obtenido a través del componente Keystone de OpenStack.

Otra posible línea de trabajo futuro sería el diseño e implementación de mecanismos capaces de soportar la modificación o borrado de componentes de la infraestructura de la red, siendo capaces de proporcionar y configurar rutas alternativas.

Sin embargo, la línea de trabajo futura más ambiciosa consistiría en mantener la conectividad en la infraestructura si se produce un fallo en algún componente y éste se cae, de manera que el resto de componentes de la federación fuesen capaces de suplir esta carencia y aprovisionar la misma conectividad que ese servicio ofrecía.

ANEXO A. Ficheros de configuración

A.1 Fichero de configuración DevStack: localrc

```
SERVICE_HOST=10.0.35.1
HOST_IP=10.0.35.1
disable_service n-net
enable_service neutron q-svc q-agt q-l3 q-dhcp q-meta ryu n-cpu n-api
FLOATING_RANGE=10.0.35.0/24
PUBLIC_NETWORK_GATEWAY=10.0.35.1
#FIXED_RANGE=192.168.0.0/24
#FIXED_NETWORK_SIZE=256

Q_PLUGIN=ryu
#Q_AGENT_EXTRA_AGENT_OPTS=(tunnel_type=gre)
#Q_AGENT_EXTRA_OVS_OPTS=(tenant_network_type=gre)
#Q_SRV_EXTRA_OPTS=(tenant_network_type=gre)

ENABLE_TENANT_TUNNELS=True

Q_HOST=$SERVICE_HOST
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST
RYU_API_HOST=$SERVICE_HOST
RYU_OFP_HOST=$SERVICE_HOST

MYSQL_PASSWORD=admin
RABBIT_PASSWORD=admin
```

```
SERVICE_TOKEN=admin

SERVICE_PASSWORD=admin

ADMIN_PASSWORD=admin


RYU_APPS=ryu.app.gre_tunnel,ryu.app.quantum_adapter,ryu.app.rest,ryu.a
pp.rest_conf_switch,ryu.app.rest_tunnel,ryu.app.tunnel_port_updater,ry
u.app.rest_quantum,ryu.app.qos_cn


# Images

IMAGE_URLS+=",https://launchpad.net/cirros/trunk/0.3.0/+download/cirro
s-0.3.0-i386-disk.img"


# Branches

KEYSTONE_BRANCH=stable/icehouse

NOVA_BRANCH=stable/icehouse

GLANCE_BRANCH=stable/icehouse

CINDER_BRANCH=stable/icehouse

HORIZON_BRANCH=stable/icehouse

NEUTRON_BRANCH=stable/icehouse


NEUTRON_REPO=https://github.com/SmartInfrastructures/neutron

NEUTRON_BRANCH=stable/icehouse

RYU_REPO=https://github.com/SmartInfrastructures/ryu.git

RYU_BRANCH=qos


NEUTRONCLIENT_REPO=https://github.com/SmartInfrastructures/python-
neutronclient.git

NEUTRONCLIENT_BRANCH=qos

LIBS_FROM_GIT=python-neutronclient
```

A.2 Ficheros de configuración ABNO con la información de cada región

```
<?xml version="1.0" encoding="UTF-8"?>
<Madrid>
        <Region>
                <ip>10.0.78.2</ip>
                <port>8080</port>
                <type>RYU</type>
        </Region>
        <Destination>
                <dest_region>Trento</dest_region>
                <tunnel>gre0</tunnel>
        </Destination>
        <Destination>
                <dest_region>Irlanda</dest_region>
                <tunnel>gre5</tunnel>
        </Destination>

</Madrid>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Trento>
        <Region>
                <ip>10.0.35.1</ip>
                <port>8080</port>
                <type>RYU</type>
        </Region>
        <Destination>
                <dest_region>Madrid</dest_region>
                <tunnel>gre1</tunnel>
```

```
        </Destination>
        <Destination>
                <dest_region>Irlanda</dest_region>
                <tunnel>gre2</tunnel>
        </Destination>

</Trento>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Irlanda>
        <Region>
                <ip>10.0.15.109</ip>
                <port>8080</port>
                <type>RYU</type>
        </Region>
        <Destination>
                <dest_region>Madrid</dest_region>
                <tunnel>gre4</tunnel>
        </Destination>
        <Destination>
                <dest_region>Trento</dest_region>
                <tunnel>gre3</tunnel>
        </Destination>

</Irlanda>
```

A.3 Fichero corregido del componente Ryu SDN: lldpy.py

A continuación se describen los cambios necesarios a realizar en el fichero:

- /ryu/ryu/lib/packet/lldp.py

del componente software RyU SDN 3.13

1) Comentar las líneas 86 y 89 del correspondiente fichero.

```
86.- # assert self.tlv_type == tlv_type
89.- # assert len(buf) >= self.len + LLDP_TLV_SIZE
```

2) Modificar el método: @classmethod de la manera que sigue:

```
126.-   @classmethod
        def parser(cls, buf):
            tlvs = []
        i=0
            while buf:
                tlv_type = LLDPBasicTLV.get_type(buf)
            i=i+1
            if tlv_type != 12:
                   tlv = cls._tlv_parsers[tlv_type](buf)
                tlvs.append(tlv)
                offset = LLDP_TLV_SIZE + tlv.len
                buf = buf[offset:]
                if tlv.tlv_type == LLDP_TLV_END:
                    break
                #assert len(buf) > 0
            if i>999:
              print "Breaking while..."
              break
            lldp_pkt = cls(tlvs)
```

```
            #assert lldp_pkt._tlvs_len_valid()
            #assert lldp_pkt._tlvs_valid()

            return lldp_pkt, None, buf

        def serialize(self, payload, prev):
            data = bytearray()
            for tlv in self.tlvs:
                data += tlv.serialize()

            return data
```

ANEXO B. Manual de usuario

B.1 Open vSwitch commands

- List all flows from the Open vSwitch

```
$ sudo ovs-ofctl dump-flows br-int
```

- Set controller from Open vSwitch

```
$ sudo set-controller br-int tcp:127.0.0.1:6633
```

- Get controller from Open vSwitch

```
$ sudo get-controller br-int
```

- Show details from Open vSwitch

```
$ sudo ovs-ofctl show br-int
```

- Add eth1 port the Open vSwitch

```
$ sudo ovs-vsctl add-port br-int eth1
```

B.2 OpenStack commands

- Create a net

```
$ neutron net-create $NET_NAME
```

This command should return the $NET_ID. In order to create a subnet in hits net and include a VM in this net, it is needed.

- Create a subnet

```
$ neutron subnet-create --no-gateway $NET_ID 10.0.4.0/24
```

- Create a VM with the name $VM_name

```
$ nova boot –image cirros-0.3.1-x86_64-uec –flavor 1 –nic
  net-id=$NET_ID,v4-fixed-ip=$IP $VM_NAME
```

This command should return the $VM_ID. In order to delete a VM or obtain specific details, it is needed.

- Get a console to the VM

```
$ nova get-vnc-console $VM_ID novnc
```

- Show all nets

```
$ neutron net-list
```

- Delete a net

```
$ neutron net-delete $NET_ID
```

- Delete a VM

```
$ neutron delete $VM_ID
```

- Pause a VM

```
$ nova pause $VM_NAME
```

- Restart a paused VM

```
$ nova unpause $VM_NAME
```

- Suspend a VM

```
$ nova suspend $VM_NAME
```

- Resume a suspend VM

```
$ nova resume $VM_NAME
```

- Reboot an active VM

```
$ nova reboot $VM_NAME
```

```
$ nova reboot --hard $VM_NAME
```

- Rename a VM

```
$ nova rename $VM_OLD_NAME $VM_NEW_NAME
```

- Examine details of network

```
$ neutron net-show $NET_NAME
```

- Examine details of subnet

```
$ neutron net-show $SUBNET_NAME
```

B.3 Ryu SDN commands

- Get the list of all switches

  GET /stats/switches

```
$ curl http://localhost:8080/stats/switches
```

- Get the description stat of the switch

  GET /stats/desc/<dpid>

```
$ curl http://localhost:8080/stats/desc/$DPID
```

- Get flows stats of the switch

  GET /stats/flow/<dpid>

```
$ curl http://localhost:8080/stats/flow/$DPID
  | python -mjson.tool
```

- Get ports stats of the switch

  GET /stats/switches

```
$ curl http://localhost:8080/stats/port/$DPID
  | python -mjson.tool
```

- Get ports description of the switch

  GET /stats/ portdesc/<dpid>

```
$ curl http://localhost:8080/stats/portdesc/$DPID
  | python -mjson.tool
```

- Add a flow entry

  POST /stats/flowentry/add

```
$ curl -X POST -d '{"dpid": "$DPID ", "match":
 {"in_port": "$PORT_IN"}, "actions":[{"type": "OUTPUT",
 "port": "$PORT_OUT"}]}'
  http://localhost:8080/stats/flowentry/add
```

- Delete all matching flow entries

  POST /stats/flowentry/delete

```
$ curl -X POST -d '{"dpid": "$DPID ", "match":
 {"in_port": "$PORT_IN"}, "actions":[{"type": "OUTPUT",
 "port": "$PORT_OUT"}]}'
  http://localhost:8080/stats/flowentry/delete
```

- Delete all flow entries of the switch

  DELETE /stats/flowentry/clear/<dpid>

```
$ curl http://localhost:8080/stats/flowentry/clear/$DPID
```

- Delete all flow entries of the switch

  POST /stats/flowentry/delete_strict

```
$ curl -X DELETE
http://localhost:8080/stats/flowentry/clear/$DPID
```

ANEXO C. ABNO API Specification

- **Create an unidirectional link between two hosts identified by their MAC addresses.**

Verb	URI
GET	`source_region=<?>&dest_region=<?>&source_mac=<?>& dest_mac=<?>&source_interface=<?>& destination_interface=<?>&operation=<?>& operation_type=<?>&id_operation'`

- **Parameter description.**

  - **Source and Destination Region** (mandatory). These variables contain the name of the regions where Ryu controllers are.
  - **Source and destination MACs of the VMs** (mandatory): the MACs from the VMs that are the origin and the destination of the LSP.
  - **Source interface** (mandatory): is the connection port of the source VM to the source Switch: the physical port that connects the source VM to the source Switch. This is necessary to correctly create the Ryu rules.
  - **Destination interface** (mandatory): is the connection port of the destination VM to the destination Switch the physical port that connects the destination VM to the destination Switch.
  - **Operation**: this string is mapped to the operation that the ABNO has to execute to configure the link. For this workflow the value is: "WLAN_ PATH__PROVISIONING".
  - **Operation_Type** (mandatory): this string is mapped to the workflow that the ABNO controller has to execute. For this workflow the value is: "*XifiWF*".
  - **ID_Operation** (not mandatory): is the identifier for the specific operation in order to trace back and reference this specific action in the network.

- **Response.**

  Returns a JSON containing the configuration parameters.

  i. Error Response Code: **400. BAD_REQUEST**.

  ii. Error Response Code: **500. INTERNAL_SERVER_ERROR**.

  iii. Normal Response Code: **200. OK**.

➢ **Example.**

- A request could be done in the following way:

```
curl 'localhost:4445?source_region=region1&destination=region3&
source_mac=00:1e:c9:bb:7e:54&dest_mac=00:13:3b:52:02:27&
source_interface=20&destination_interface=20&operation=
WLAN_PATH_PROVISIONING&Operation_Type=XifiWF&ID_Operation=1234'
```

- And the response could be something like that:

```
{
      "ID_Operation":" 1234"
      "Operation Type":"XifiWF",
      "Result": "LINK_CONFIGURED"
      "Error Code": "NO_ERROR"
}
```

REFERENCIAS

[1] A. Akella, D. Maltz, and T. Benson, "Unraveling the complexity of network management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, Barkeley, CA, USA, 2009, pp. 335-348.

[2] S. Schenker. (2011) The Future of Networking and the Past of Protocols. [Online]. 2015: http://www.youtube.com/watch?v= YHeyuD89n1Y

[3] FI-PPP. (2013) The Future Internet Public-Private Partnership. [Online]. http://www.fi-ppp.eu

[4] J. González, F. Álvarez, L. Contreras, and O. González, "Inter-domain Monitoring and Software-Defined Network Connectivity for Federated Infrastructures Management," , Spain.

[5] E. Escalonay, J. Aznar, L. Contreras, O. González, G. Cossu, E. Salvadori, and F. Facca, "Using SDN for doud services provisioning: The XIFI use-case," in *XIFI*.

[6] IEEE. (2005) 802.1 Q - Virtual Bridged Local Area Networks. [Online]. http://www.ieee802.org/1/pages/802.1Q.html

[7] T. Koponen, S. Shenker, A. Singla, B. Raghavan, and J. Wilcox A. Ghodsi, "Intelligent design enables architectural evolution," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, New York, NY, USA: ACM, 2011, pp. 3:1-3:6.

[8] T. Koponen, S. Shenker, A. Singla, B. Raghavan, and J. Wilcox A. Ghodsi, "Software-defined internet architecture: Decoupling architecture from infrastructure," in *roceedings of the 11th ACM Workshop on Hot Topics in Networks*, New York, NY, USA: ACM, 2012, pp. 43-48.

[9] IEEE. (2006) 802.1 ad - Provider Bridges.

[Online]. http://www.ieee802.org/1/pages/802.1ad.html

[10] IEEE. (2008) 802.1 ah - Provider Backbone Bridges.

[Online]. http://www.ieee802.org/1/pages/802.1ah.html

[11] Open Network Foundation. (2012) Software-Defined Networking: The New Norm for Networks. [Online].

https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[12] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," in *Communications Magazine, IEEE, vol. 51, no. 7*, 2013.

[13] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig, "Software-Defined Networking: A Comprehensive Surve," , 2014.

[14] L.M. Contreras, P.A. Aranda, G. Gardikis, and G. Darladimas, "Technical Note TN1: Cloud networking techniques and technologies review," , 2014.

[Oline]. https://artes.esa.int/projects/doudsat

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," in *SIGCOMM Comput. Commun. Rev., vol. 38, no. 2*, 2008, pp. 69-74.

[16] Open Networking Foundation. (2014) . [Online]. https://www.opennetworking.org/

[17] H. Yin, H. Xie, T. Tsou, D. Lopez, P. A. Aranda, and R. Sidi. (2013) SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. [Online]. http://tools.ietf.org/html/draft-yin-sdn-sdni-00

[18] XIFI. XIFI proyect. [Online]. https://fi-xifi.eu/home.html

[19] FIWARE. FIWARE Open APIs for Open Minds. [Online]. http://www.fiware.org/

[20] A. Farrel and D. King. (2015, March) A PCE-Based Architecture for Application-Based Network Operations. [Online]. https://tools.ietf.org/html/rfc7491

[21] XIFI. (2013) D3.1: XIFI infrastructure adaptation components API open specification. [Online]. https://bscw.fi-xifi.eu/pub/bscw.cgi/d44705/XIFI-D3.1-XIFI_infrastructure_adaptation_components_API_open_specification.pdf

[22] The OpenNaaS Community,. (2013) OpenNaaS. [Online]. http://www.opennaas.org/

[23] D. Wilson. (2013, August) OpenNaaS and Virtual CPE: a new way to connect to the Internet. [Online]. http://www.terena.org/publications/tnc2013-proceedings/

[24] Mantychore. Mantychore Project. Provide infrastructure resources and IP networks as a service. [Online]. http://www.mantychore.eu/

[25] XIFI. (2015) D3.7: XIFI Infraestructure Network Adaptation Mechanisms API. [Online].

https://bscw.fi-xifi.eu/pub/bscw.cgi/d96823/XIFI-D3.7-XIFI_Infrastructure_Network_Adaptation_Mechanisms_API_v2.pdf

[26] A. Farrel, J.P. Vasseur, and J. Ash, "A path computation element (PCE)-based architecture," in *IETF RFC 4655*, 2006, pp. 1-40.

[27] OpenStack. Open Source Cloud Computing Software. [Online]. https://openstack.org/

[28] OpenStack Architecture. [Online]. http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html

[29] XIFI. (2014) D3.4: XIFI Infraestructure Network Adaptation Mechanisms API. [Online]. https://bscw.fi-xifi.eu/pub/bscw.cgi/d64447/XIFI-D3.4-XIFI_Infrastructure_Network_Adaptation_Mechanisms_API.pdf

[30] Project Floodlight. OpenSource Software for Building SDN. [Online]. http://www.projectfloodlight.org/floodlight/

[31] RYU. RYU SDN Framework. [Online]. http://osrg.github.io/ryu/

[32] RYU GitHub. (2015, May) Ryu component-based software defined networking framework. [Online]. https://github.com/osrg/ryu

[33] RYU Project Team. RYU SDN Framework. [Online]. https://osrg.github.io/ryu-book/en/Ryubook.pdf

[34] OpenStack. DevStack - an OpenStack Community Production. [Online]. http://docs.openstack.org/developer/devstack/

[35] Mininet. Mininet. An Instant Virtual Network. [Online]. http://mininet.org/

[36] Mininet. Mininet Walkthrough. [Online]. http://mininet.org/walkthrough/